80-FM-47

JSC-16794

User's Guide for the Flight Design System (FDS)

Mission Planning and Analysis Division

August 1980

NASA
National Aeronautics and Space Administration

Lyndon B. Johnson Space Center
Houston, Texas

SHUTTLE PROGRAM

USER'S GUIDE FOR THE FLIGHT DESIGN SYSTEM (FDS)

By H. Rudy Ramsey, Michael E. Atwood, William G. Frisius,
Althea A. Turner, and John K. Willoughby,
Science Applications, Inc.

Approved: _____
Kenneth A. Young, Chief
Flight Planning Branch

Approved: _____
Ronald V. Berry, Chief
Mission Planning and Analysis Division

Mission Planning and Analysis Division

National Aeronautics and Space Administration

Lyndon B. Johnson Space Center

Houston, Texas

August 1980

Page intentionally left blank

FOREWORD

This document is the user guide for the Flight Design System (FDS), an interactive aid for trajectory, attitude, and consumables planning of Space Transportation System missions. FDS has been developed by the Mission Planning and Analysis Division, NASA Lyndon B. Johnson Space Center, Houston, Texas, and by its contractors. This user guide was developed for MPAD by personnel of the Man-Computer Systems Division, Science Applications, Inc., Englewood, Colorado, under contract NAS9-15535.

FDS has two major components. "Subsystem X" is a rapid, interactive planning system residing primarily on an Interdata 8/32 computer, while "Subsystem Y" is a batch-only, high-fidelity simulation system residing on a Univac 1108. This present version of the user guide contains an overview of the entire FDS system, but contains detailed information only on Subsystem X. When Subsystem Y is better defined, it is intended that this user guide be augmented with more detailed information on that portion of FDS.

This document was developed in an unusual manner. SAI analysts first compiled a complete list of the identifiable concepts which must be understood for successful use of FDS. The predecessor-successor relationships among these concepts were then identified. The resulting, partially ordered set of concept descriptions might loosely be called a "semantic network." On the basis of the content and structure of this network, the concepts were then divided into "modules", which were developed into actual document sections via a technique called "storyboarding". Further information on the approach used can be obtained from the authors. Constructive criticisms of the document content and/or format would be appreciated by both SAI and MPAD.

The authors wish to acknowledge the assistance of many MPAD personnel in the definition of the material contained in this document. Dave Alexander, the project monitor, assisted in a variety of ways, and helped particularly with the design of the example problem. Bob Davis and George Weisskopf critiqued early versions of the user guide semantic network and draft user guide sections, and provided considerable assistance in understanding the detailed behavior of the FDS. Bernie Roush provided information and demonstration of the document processing capabilities of FDS. Sandy Price and Jim Cook helped convert the example problem scenario to a specific solution plan useable for example discussions in the design guide. Others, too numerous to mention, helped resolve specific questions, demonstrated capabilities, criticized drafts, and provided other kinds of assistance.

Page intentionally left blank

TABLE OF CONTENTS

Page intentionally left blank

USER'S GUIDE

for the

FLIGHT DESIGN SYSTEM

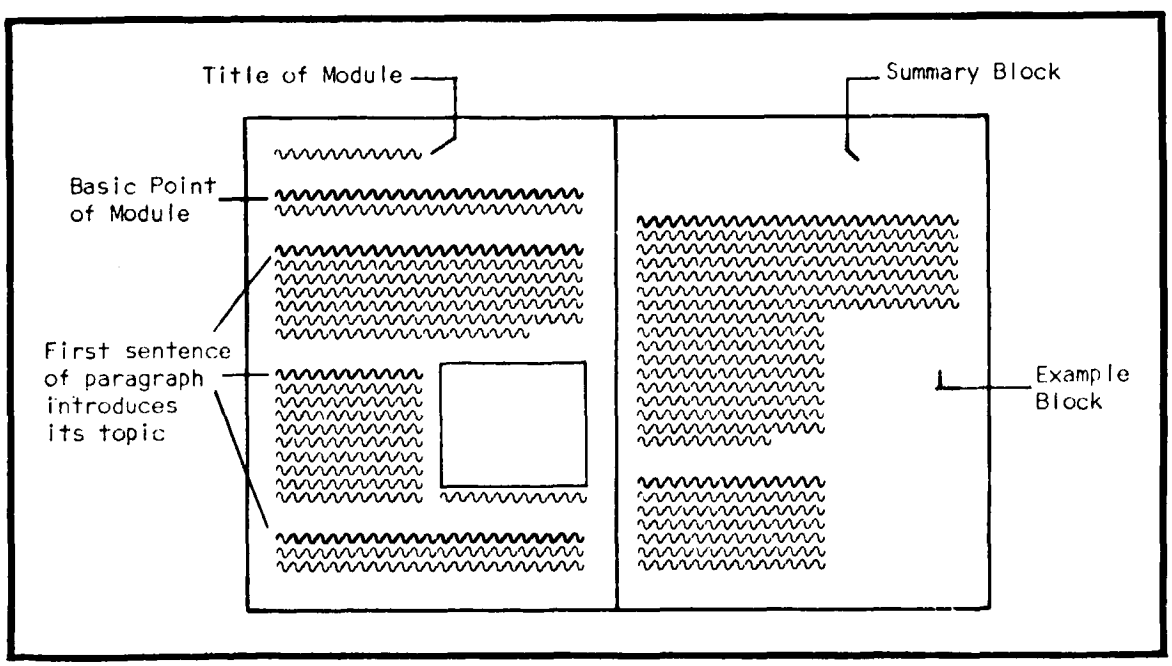
(FDS)

01 INTRODUCTION TO THE FLIGHT DESIGN SYSTEM USER GUIDE

This User Guide is designed as an aid to the community of users of the Flight Design System (FDS).

The FDS User Guide has two purposes: (1) to provide material which aids in user understanding of FDS and its environment, and (2) to serve as a reference manual for the use of FDS in flight design. The first section of this document presents information about the Flight Design System in the context of flight design as carried out by NASA. It also introduces the FDS user to the structure of FDS and to constructs within FDS (such as files of information or the part of FDS which interacts directly with the user). The second section of this document is a guide to the commands available to FDS users. In addition, a glossary of important terms, an index to terms, and a quick reference to the commands of FDS are included in the Guide.

The layout of this document provides its user with easy access to desired information. Each topic is presented in a two-page "module". Each of these modules is introduced by an appropriate sentence which specifies the basic point of the module. Each paragraph within a module begins with a sentence which introduces the main topic of the paragraph. Specialized types of information, such as a summary or an example problem, are presented with a module to aid in quick review (or in finding a module on a related topic). Within each module, important concepts which are introduced there for the first time in the User Guide are printed in capital letters.

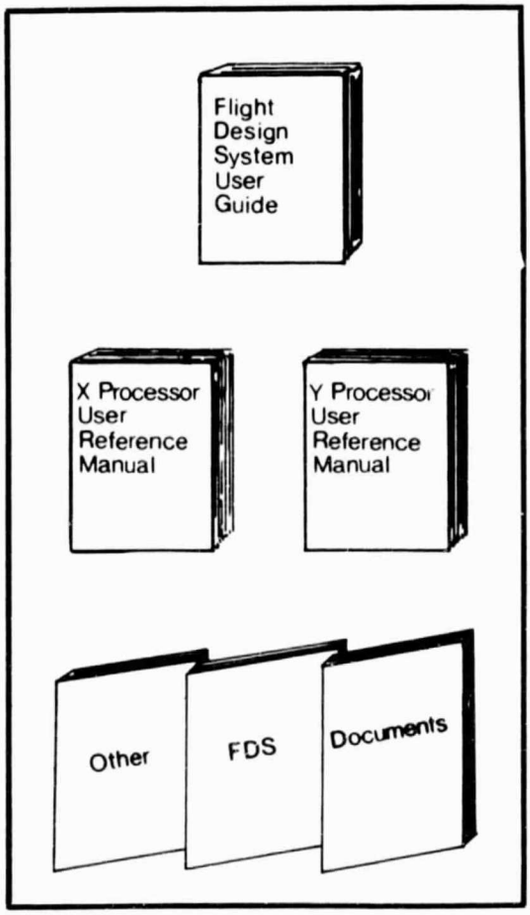


1-1. The layout of the FDS User Guide helps users to access desired information.

Each module of the FDS User Guide contains a summary block to aid in quick reference of material contained in that module. This block is highlighted like this, with a red background. It is always located at the top of the right hand page of a module. A user who does not wish to read a module in detail may use this block as a quick review of the module.

Supplementary information is provided by a glossary, an index, and a quick reference page of FDS commands. Definitions of important terms can be referenced easily in the glossary section. An index of the terms used in the User Guide provides page numbers on which the terms are discussed. The page numbers on which major definitions or explanations appear are underlined. A page of sample commands, showing basic forms, is located in the back of this document, along with some aids to finding desired commands.

The FDS User Guide is aimed at a community of users of varying background who are all engaged in flight design using FDS as a tool. Obviously, not all users will access this document in the same way. Users who already know information in a module may wish to simply read the summary block or to skip that module altogether. New users may wish to read the modules carefully and in order the first time through. All users should be able to use this document to become more proficient in their use of FDS.



1-2. Several documents serve as reference material for FDS use.

Several other documents are available which are more detailed references to particular aspects of FDS. The "X Processor User Reference Manual" provides detailed information about the X Processors. Included in this volume is a guide to the documentation function of FDS. The "Y Processor User Reference Manual" is a reference manual to the various high fidelity simulation processors available on FDS. For users who desire a deeper understanding of FDS, the "FDS-2 Software Design and Description" (79-FM-11) is recommended. Most users will not need the level of detail presented there, however. Finally, hardware manuals are available for the various computers, terminals, etc., which make up FDS. For most users, though, this User Guide and the processor reference manuals should be an adequate source of information.

An example problem has been selected which is carried though the Guide. The example is printed on a green background for easy identification. Not all modules have an example problem block. By reading through the example blocks from front to back, the reader can obtain a review of the procedures involved in FDS use.

02 OVERVIEW OF FLIGHT DESIGN

Flight design is a complex, iterative process involving several organizations within NASA.

Flight Design for the Space Transportation System (STS) is a planning process. For each flight this process produces a schedule of events that will occur between the lift-off and landing of the Space Shuttle. The planned events must satisfy the mission objectives and be operationally feasible. That is, they must stay within the performance capabilities of the spacecraft, its crew, and the ground support system. Several kinds of analysis and planning are used to produce a flight design. Trajectory analysis specifies launch criteria, orbital maneuvers and other activities that are needed to guide the spacecraft. Attitude planning determines how the spacecraft is to be pointed so that sensors and communication equipment can function properly and the temperatures can be controlled. Consumables analysis is used to assure that the consumption of energy, propellants and environmental control chemicals is within the limits of the spacecraft. Crew activity considerations can cause adjustments to the plan in order to assure the effectiveness and well being of the crew members.

2-1. Space shuttle missions require precise planning.

Virtually all decisions made in considering one aspect of flight design affect the others. For example, trajectory events can drive attitude, crew and consumables planning, or crew activities constraints may necessitate trajectory alterations. Because of this interaction, flight design is fundamentally a process of fitting together several different specialty-oriented views of mission activities so that things happen in a feasible order and that the spacecraft, ground support systems, and crew can function effectively. The integration of the various technical aspects needed to accomplish flight design requires the ability to iterate, that is, to produce preliminary or tentative flight designs and then refine or redo those designs until all deficiencies are removed.

Flight design is a planning process that uses various technical inputs to produce flight plans and profiles for the Space Transportation System. The process begins with a request from SPIDPO to MPAD for assistance in developing a Payload Integration Plan. From this point, MPAD directs the iterative process of flight design from its initial stages of the development of a Conceptual Flight Profile, through a review process involving SPIDPO and CTPD, to the final production of the detailed Operational Flight Profile.

Organizational responsibilities and communication mechanisms have evolved in order to accomplish this iterative integration task. The flight design process begins with a request from the Shuttle Payload Integration and Development Program Office (SPIDPO) to the Mission Planning and Analysis Division (MPAD) for assistance in developing a Payload Integration Plan (PIP). The cargo for the flight has already been determined. Thus, the objectives for the flight are already established. From this starting point, MPAD develops a Conceptual Flight Profile (CFP). This CFP is a preliminary attempt to produce a flight design that satisfies the integrated constraints from all the technical concerns. It is also the basic instrument for iteration. The various organizations within MPAD, the Crew Training and Procedures Division (CTPD), and SPIDPO are the principal participants in the review of the Conceptual Flight Profile. A Cargo Integration Review meeting and the publication of a final Conceptual Flight Profile end this phase of flight design.

INVOLVEMENT OF GROUP OR ORGANIZATION IN FLIGHT DESIGN				
CUSTOMER	SPIDPO	MPAD	CTPD	MISSION PLANNING PHASE
X	X			Determine payload specifications
	X			Assign payload to flight
	X	X		Develop Payload Integration Plan
		X		Develop Conceptual Flight Profile
X	X	X	X	Review Conceptual Flight Profile
		X		Develop Operational Flight Profile

2-2. The flight design process involves several organizations within NASA.

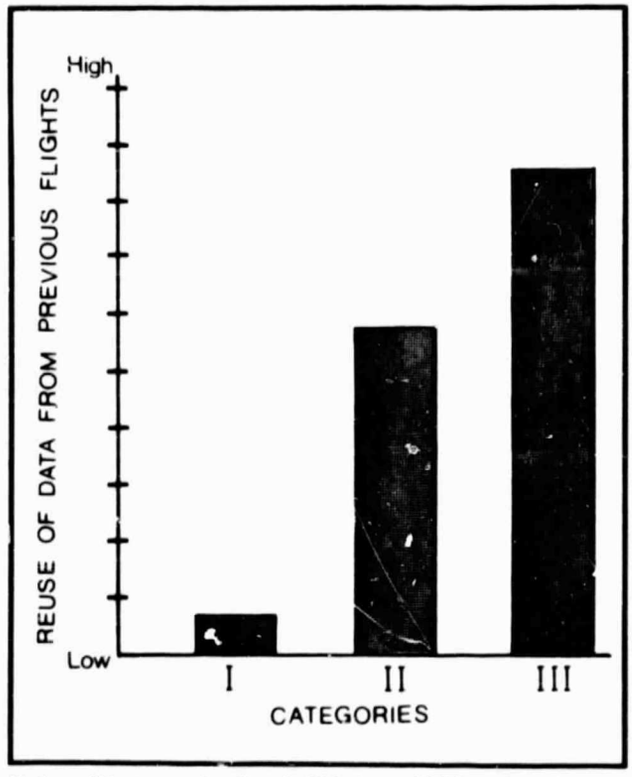
The CFP is the input to the development of a detailed crew activity timeline by the Crew Training and Procedures Division. Iteration with MPAD occurs during this process. Finally, about one year before the flight, the development of an Operational Flight Profile (OFP) begins. This is used for flight simulations and for support during the actual flight. Approximately five months before the flight, the flight design is completed or "frozen" and the final OFP is published.

03 THE FLIGHT DESIGN SYSTEM

The Flight Design System helps in the planning and design of shuttle missions as well as providing documentation and other support necessary in the development process.

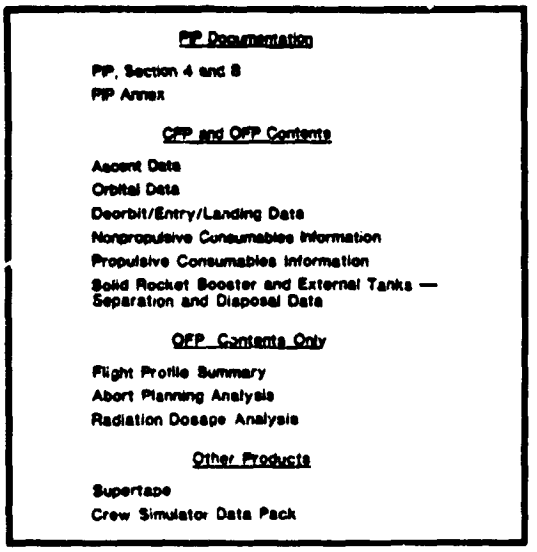
MPAD uses the computerized FLIGHT DESIGN SYSTEM (FDS) in support of the flight design process. The most basic requirement of the Flight Design System is to provide a manpower-saving technique to plan essentially repetitive flights at a density of up to 20 flights per year. As the number of planned flights increases, and similarities between flights become evident, operational envelopes will be standardized for specific payloads and experiments. For each new flight, a review of previous flights will be performed to determine if any of them were similar enough to the proposed flight that they may be used as a model. Using the flight requirements and a model flight (if it exists) as a starting point, a Conceptual Flight Profile is generated which meets the flight criteria. It is entirely possible that several flight designers will be working concurrently on the same flight or will be using the same model timeline to design different flights.

If the proposed flight design does not require the development of new flight design techniques, the flight is categorized based on the procedures and amount of work involved in generating the initial profile (CFP). CATEGORY I flights are the highest level of complexity that will be supported by FDS. Flights in this category require a large amount of new flight planning with only limited use of standard event timelines that were developed for earlier flights. CATEGORY II flights will typically be a near repeat of a previous flight, involving only minor changes to event timelines and a limited amount of additional analysis. The approach to CATEGORY III flights will consist essentially of drawing information from a data base library -- which contains previous flight data -- changing required values, and re-running the flight profile on the FDS. Flights more complex than Category I (Category 0) involve the development of new techniques or forms of analysis. While FDS is not designed to support such flights, the system can still serve as a useful tool in developing new procedures.



3-1. "Categories" of flights differ in the degree to which they make use of data from previous flights.

FDS is a computer system that aids in mission planning and flight design. FDS supports the flight design process by providing interactive simulations, documentation such as: PIP Section 4 and 8, PIP Annex, CFP, and OFP, and products such as: Supertape, and Crew Simulator Data Pack. FDS is designed to save manpower in repetitive planning, as well as provide tools necessary to design new or unique flights.



3-2. FDS assists in the generation of several documents and data products.

<u>FDS is used as a planning aid throughout the flight design process</u>. First the CFP is produced. The CFP is based on a relatively low-fidelity FDS simulation from lift-off to touchdown. After the CFP is generated, it will be examined for conflicts in various aspects of the flight. Then it will be refined in further iterations via interactive FDS sessions until all major criteria specified in the PIP are satisfied. Following publication of the CFP, the associated data files will be archived. After the CFP is reviewed, additional refinements of the flight profile may be undertaken. After the final refinements are made, FDS is used to begin preparing the OFP. The OFP represents a high-fidelity FDS simulation of an entire shuttle mission from lift-off to touchdown. After the OFP is planned, produced, and published, a <u>SUPERTAPE</u> and, for the first time, a <u>CREW SIMULATOR DATA PACK</u> are produced by FDS. These products are used by CTPD to support the generation of the crew activities timeline. Any conflicts or discrepancies discovered after publication of the OFP will necessitate iterations of all or part of the CFP-OFP development, planning, and publication cycle.

<u>FDS aids in document production throughout the design process and will save flight design data for future use</u>. A flight designer may generate all or part of the FDS-supported document products by obtaining a copy of any standardized, skeletal document from the special-purpose document processor (Xerox) and merging it with data generated using FDS. The user may verify the merged text at an FDS terminal before final processing. As a last step in the flight profile development process, the timelines and parameters used by FDS to generate the delivered products, along with any supportive information, will be archived as a history file. This information may be used as a model for future planning.

<u>In summary, MPAD uses the FDS in flight design activity from generation of PIP requirements (long before liftoff) until the flight is actually flown</u>. FDS provides the computer simulations that support the flight design process. The FDS user performs the decision process and FDS performs repetitive analysis, math-model execution, standardized documentation, and fine tuning necessary to complete the flight design process.

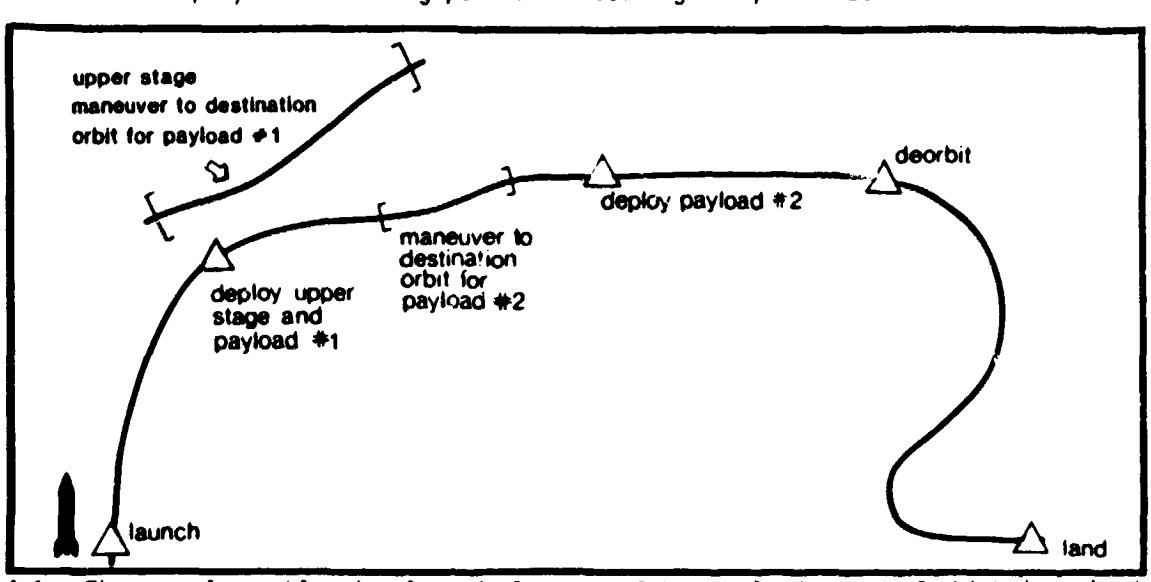
04 AN EXAMPLE TO ILLUSTRATE FDS USAGE

An example problem has been selected to illustrate the use of the FDS.

Most of the basic concepts discussed in the User Guide are also illustrated with a single, concrete example problem. This problem is described in this section, and is used throughout the document. Each discussion of the example problem is presented on a green background. The reader who understands the example problem in general terms should have the background necessary to comprehend the example discussions which are distributed throughout the User Guide.

Although the example mission is not as complex as a typical STS mission, it contains realistic elements that frequently appear and that are appropriate to illustrate the use of FDS. The objective of the example mission is to deploy two payloads. The first payload is a satellite which must be placed in a geosynchronous orbit -- that is, in an orbit in which the satellite will remain above a particular point on the earth's equator. The Orbiter is not capable of taking the payload to that orbit using its own orbital maneuvering system. Therefore, an upper stage is attached to the payload and will be deployed from the orbiter's cargo bay. That upper stage will perform a transfer maneuver from the Orbiter's parking orbit to the satellite's final geosynchronous orbit. After the deployment of the upper stage and its payload, the Orbiter will maneuver to a second orbit and deploy the second payload.

The elements of FDS that are used in the generation of a flight design are, of course, not unique. The user is free to choose the processors to be used, the order in which they will be involved, the names of tables employed, and the operating modes of FDS. No unique problem-solving sequence can be associated with a given problem. Nevertheless, a representative problem-solving process has been assumed for the example problem in order to illustrate FDS capabilities and features. The reader should not assume that the solution illustrated represents the only way the FDS could be used. Experience with the system will enable the user to become increasingly more efficient in the exploitation of the FDS capabilities. This is particularly true as results from previously-solved problems are employed as starting points for solving new problems.



4-1. The example problem involves deployment of two payloads, one of which is raised to a geosynchronous orbit by an upper stage maneuver.

A single example problem is used for illustrative purposes throughout the User Guide. Discussions of the example problem appear on a green background. The problem involves two payloads, one of which is to be placed in a geosynchronous orbit.

Some simplifying assumptions have been made to insure that the problem is a manageable example. For example, it is assumed that the launch date and time have been determined previously. Ordinarily, a launch window analysis would be performed to determine these parameter values. It should also be noted that the order in which planning is done for the upper stage and for the Orbiter, after the first payload deployment, is entirely arbitrary.

PAYLOAD 1

TARGET ORBIT (GEOSYNCHRONOUS):
altitude = apogee = perigee = 19,323 nm.
inclination = 0 (equatorial)
eccentricity = 0 (circular)

SATELLITE WEIGHT AT DEPLOYMENT = 12,542 lbs.

PAYLOAD 2

TARGET ORBIT:
altitude = 200 nm.
inclination = 28.5 degrees
eccentricity = 0 (circular)

WEIGHT = 6,320 lbs.

LAUNCH:
DATE = 23 April 1981
GREENWICH MEAN TIME = 10:36:00
LOCATION = Eastern Test Range
MECO ALTITUDE = 57 nm.
LIFT-OFF WEIGHT = 4,504,485 lbs.

LANDING:
LOCATION = Eastern Test Range
CONSTRAINTS = DAYLIGHT (30 minutes before sunset, after sunrise)

INITIAL PARKING ORBIT
altitude = apogee = perigee = 150 nm.
inclination = 28.5 degrees

FIRST SEPARATION MANEUVER
velocity delta-V = 50 fps.
engine(s) = Foreward RCS

SECOND SEPARATION MANEUVER
velocity delta-V = 2.5 fps.
engine(s) = Aft RCS

4-2. The example problem uses simple, fairly common orbits and maneuvers.

05 DETAILS OF THE EXAMPLE PROBLEM AND SOLUTION

To help the reader understand the illustrative material presented elsewhere, details about the example problem and the assumed solution process are presented here.

This example mission is relatively simple and involves several aspects that are likely to have been designed previously or which are similar to past missions. The flight designer would probably use data and procedures that he or she had stored for the purpose of future reference. In the illustrations which follow in this book, the assumption is sometimes made that computer files exist which will be retrieved for the purpose of review and modification. At other times, it is assumed that the designer will "start from scratch". Although these two problem-solving modes are assumed arbitrarily in order to best illustrate FDS capabilities, it is also true that real world flight design will be a mixture of redesign of previous solutions together with original design efforts.

A normal but not unique sequence in the flight design process begins with basic trajectory design. After launch window analysis and the selection of a launch date, the conditions at orbital injection are determined. The event that is used as the beginning of the orbital phase of the flight is called Main Engine Cut-Off (MECO). From MECO, the orbital maneuvers are laid out. This process proceeds until the approximate deorbit conditions can be determined. An analysis of landing opportunities leads to the selection of the orbit on which the deorbit burn will be made.

After the basic trajectory events are determined, the design of the attitude profile can be accomplished. Crew requirements are factored into both the trajectory and attitude profile design. The completion of the attitude profile design allows an analysis of the propulsive consumables to be done. The non-propulsive consumables profiles can also be determined. Any violations of constraints or recommendations for more efficient use of the spacecraft or crew can cause the events of the flight design timeline to be altered.

The completion of the basic timelining, of events pertaining to trajectory, attitude profiles, basic crew activity, and consumables management are collected and documented in the Conceptual Flight Profile (CFP). After approval cycles are completed, the detailed Operational Flight Profile (OFP) is generated in approximately the same sequence. The Flight Design System is used throughout the process of producing the CFP and the OFP.

One approach to the solution of the example problem begins with trajectory design. Trajectory design involves a series of somewhat sequential planning steps, which correspond to the planned trajectory events. Once a satisfactory trajectory design has been established, attitude and consumables planning can be initiated.

TRAJECTORY ANALYSIS AND DESIGN

ORBITER THROUGH FIRST PAYLOAD DEPLOYMENT

Ascent Trajectory

Staging

1st burn of Orbital Maneuvering System (OMS) to achieve parking orbital altitude

2nd burn of OMS to circularize parking orbit

Deploy 1st payload

Perform separation maneuver with Reaction Control System (RCS) to move orbiter away from 1st payload

FIRST PAYLOAD AND ITS UPPER STAGE

Perform maneuver to change orbital plane and raise orbital altitude

Perform maneuver to make remainder of orbital plane change and to circularize in geosynchronous orbit

ORBITER AFTER FIRST PAYLOAD DEPLOYMENT

Perform OMS maneuver to achieve orbital altitude needed for second payload

Perform OMS maneuver to put shuttle in circular orbit for deployment of second payload

Deploy second payload and perform separation maneuver

Deorbit

Groundtrack check (orbiter branch of trajectory)

ATTITUDE ANALYSIS & DESIGN

PROPULSIVE CONSUMABLES ANALYSIS & DESIGN

NON-PROPULSIVE CONSUMABLES ANALYSIS & DESIGN

PUBLICATION OF CONCEPTUAL FLIGHT PLAN

DETAILED SIMULATION AND ANALYSIS

PUBLICATION OF OPERATIONAL FLIGHT PLAN

5-1. There are a number of planning operations involved in the solution of the example problem.

06 FUNCTIONAL AREAS OF FDS

FDS consists of four main components: (1) X Processors, (2) Y Processors, (3) document processing system and (4) an Executive program.

There are two types of application processors, the X Processors and the Y Processors, that are designed to perform specific flight design tasks. The application processors may be executed individually or in sequence to perform part, or all, of a flight design task. The most basic requirement for X Processors is that the fidelity be sufficient for producing the Conceptual Flight Profile (CFP) and establishing the feasibility of the Operational Flight Profile (OFP). Y Processors generate simulations and actually produce the data products required for OFPs.

X PROCESSORS consist of a library of independent application processors that are used as building blocks in flight design. These processors may be executed in interactive or batch mode and in whatever sequence is necessary to satisfy the computational requirements of a particular flight design task. That is, the user essentially structures a special-purpose program by the proper selection and sequencing of processors. The X Processors are designed to perform computational routines in an interactive mode. The fidelity of these processors is consistent with CFP development, but may be insufficiently detailed for OFP development. The X Processors are sometimes collectively called "Subsystem X".

X PROCESSORS
ATTITUDE & POINTING
CONSUMABLES & MASS PROPERTIES
MANEUVER - TARGETING
PROPAGATION
SHUTTLE UPPER STAGE
TRAJECTORY/AUXILIARY DATA SUPPORT
UTILITY
CONTROL, DISPLAY, & SIMULATION

6-1. Listing of types of X Processors.

The purpose of Y PROCESSORS is to generate high fidelity simulations of flight profiles and to produce the data products required for OFPs. Typically, the input parameters for Y Processors will consist of flight profile data generated on X Processors and transferred by the FDS user to Y Processors, through the FDS Executive. Y Processors reside on another computer, but are considered part of FDS. The Y Processors are sometimes collectively called "Subsystem Y".

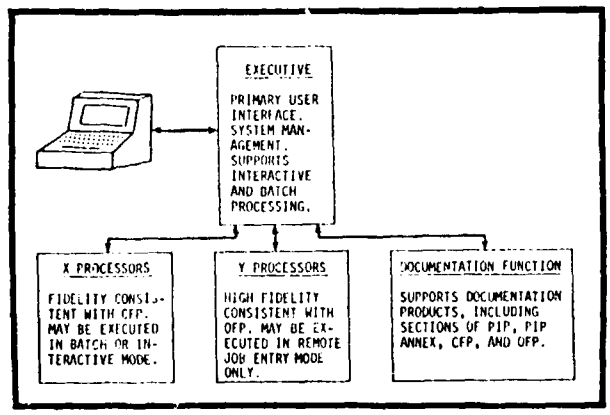
Y PROCESSORS
ASCENT SIMULATIONS
ON-ORBIT SIMULATIONS
DEORBIT-TO-LANDING SIMULATIONS
RETURN TO LAUNCH SITE SIMULATIONS
PROXIMITY OPERATIONS SIMULATIONS
PROPULSIVE CONSUMABLES-USAGE SIMULATIONS
SUPERTAPE GENERATION
CREW SIMULATOR DATA PACKAGE GENERATION
SPECIAL DISPLAYS

6-2. Listing of types of Y Processors.

FDS has four components: X Processors, Y Processors, a documentation system, and an Executive program. X Processors are executed in interactive or batch mode and are used primarily for conceptual flight design. Y Processors are executed in batch mode only, reside on another computer, and are used for very detailed simulation and design. The documentation system is used to create formal documentation, such as CFP and OFP documents. The Executive interfaces the user to the above functional areas. It provides capabilities for data storage and retrieval, editing, and processor execution.

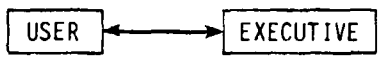
There are also some special purpose applications of FDS, primarily document production. The FDS user generates data that will be used in CFPs and OFPs. A special purpose processor is provided that allows the user to merge this information with the supporting text of the CFP and OFP documents.

The EXECUTIVE interfaces the user to the other functional areas and is the functional area of FDS that processes and responds to all user commands. The FDS Executive is designed to support interactive as well as batch processing with X Processors, to initiate remote jobs with Y Processors, and to allow access to document preparation facilities. The Executive provides the user with functional capabilities in three areas: data storage and retrieval, editing, and processor execution. These functions enable the user to get ready to use the application processors and provide the user with access to master data bases and the ability to retain data from one FDS session to the next. The input and output associated with the processors that perform flight design tasks and the order in which these processors are executed is usually prescribed by FDS tables. The execution control function is used to control the execution of processors.



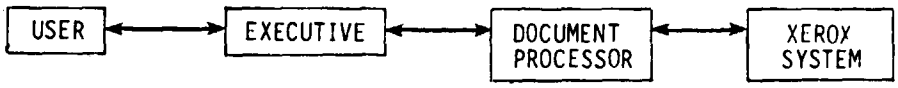
6-3. The executive processes and responds to user commands and interfaces to other functional areas.

To solve the CFP portion of the example problem using FDS, the user must first prepare tables necessary to execute the particular X Processors required in the flight design. To do this, the user interacts with the Executive.



The X Processors are then executed, again using the Executive, through which all user interaction occurs.

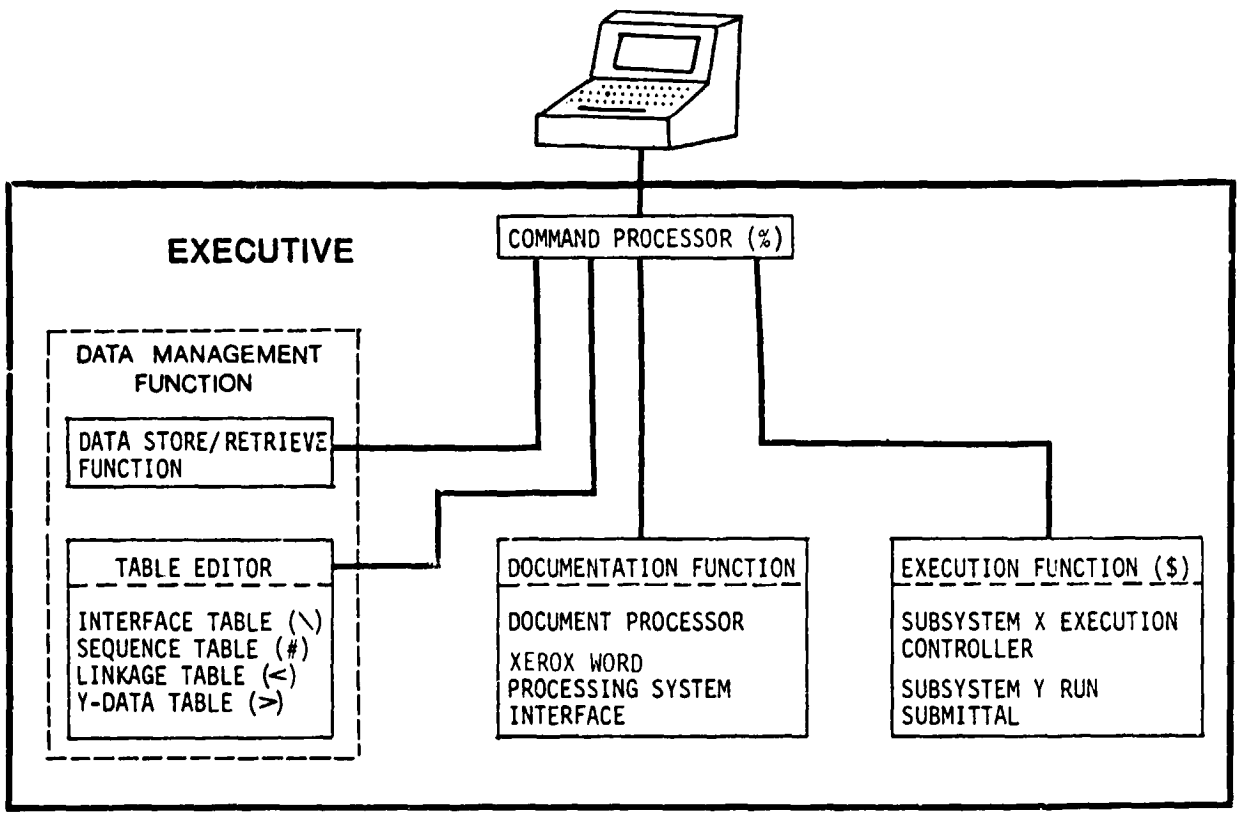
When all necessary planning iterations have occurred, the document processor is executed to produce the CFP document.



07 THE EXECUTIVE INTERACTS WITH THE USER

The Executive interacts with the user and processes the commands that are used to do flight design tasks.

The EXECUTIVE is the component of the FDS that interacts with the user and allows access to the other components. The Executive provides users with a single interface to these various components. Through the Executive, the user can access the X Processors, the Y Processors, and the documentation system, and cause data to be transferred among these components. The Executive performs four functions for the user -- data management, processor execution, command processing, and access to the documentation function.



7-1. There are four basic functions performed by the FDS Executive.

The DATA MANAGEMENT FUNCTION is concerned with creating, deleting, storing, accessing, and modifying files. In performing a flight design task with the FDS, users will access files that contain standard flight profiles, standard processing sequences, flight requirements, etc. Files will be created to store the flight profile, and other associated data, for the current flight design task. These activities are handled by the data management function.

The FDS Executive consists of several components. All user commands are processed by the Command Processor, which determines which of the components to invoke. The Data Management function provides the capability to access, create, delete, store, and modify files. The Execution Function controls the execution of X and Y Processors. For producing documentation products, the Documentation Function is used.

DATA MANAGEMENT CAPABILITIES

- Deleting, creating, accessing, and editing personal files for permanent retention of user data from session to session
- Access to master data bases
- Entering and storing X Processor input and output data
- Managing (naming, storing, deleting, etc.) X Processor output data
- Entering and storing Y Processor input and output data
- Managing Y Processor output data

EXECUTION FUNCTION CAPABILITIES

- Control the execution of X Processors
- Submit jobs which execute Y Processors

DOCUMENTATION FUNCTION CAPABILITIES

- Combine flight design data with supporting text
- Prepare CFPs, OFPs, and parts of the PIP documents

7-2. Each component of the FDS Executive performs certain tasks.

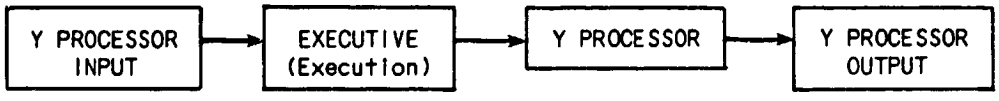
The EXECUTION FUNCTION of the Executive controls the execution of the processors. This function has two principal parts -- the X Processor execution controller and the Y Processor run submittal controller. The X Processor execution controller can operate in one of four modes -- one processor at a time (manual), automatic, semiautomatic, or batch. The Y Processor run submittal controller causes runs using Y Processors to be processed in the remote-job-entry (RJE) mode.

The COMMAND PROCESSOR provides access to other functional components of the Executive. All user commands are processed by the Command Processor, which then determines which component to invoke. The Command Processor prompts the user with the % symbol, after which the user inputs a command. There is a unique prompt symbol associated with each component of the FDS, and the prompt symbol changes to indicate which component of the FDS is active.

The DOCUMENTATION FUNCTION provides a means for input, editing, and output of text documents. These documents include OFPs, CFPs, and the Flight Design Annex to the Payload Integration Plan (PIP). Skeletal forms of these documents are stored on the Xerox word processing system. The FDS Executive provides a means to merge flight design data with these skeletal forms to produce final documents.

In solving the OFP portion of the example problem with FDS, the user will make use of the Y Processors. The Y Processor job is prepared by setting up an input file.

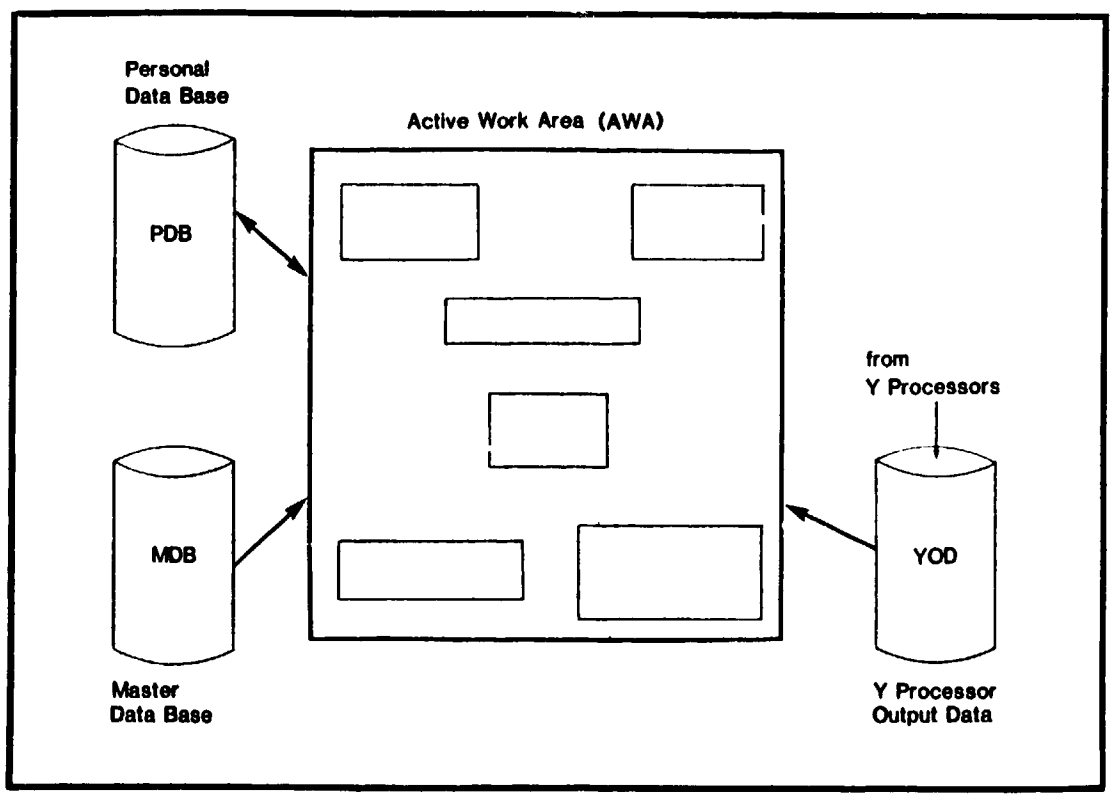
The job is then transmitted, via the Y Processor run submittal controller, to a separate computer on which the Y Processors reside. The job may or may not actually be run for several hours. When the job has run, the output is returned to FDS and placed in a file. When the user next signs on to FDS, the output is available for reading and editing.



08 DATA STORAGE AREAS WITHIN FDS

There are four major data storage areas over which the user has varying degrees of control.

The ACTIVE WORK AREA (AWA) is one of the several areas used by FDS to store data. The AWA consists of random access memory and disk storage, and is automatically allocated to the user at sign-on. Many files may be stored in the AWA. The AWA stores input data for use by the executing processors, and is used for constructing and manipulating data and tables. In addition, all retrievable output data from X Processors are stored there. The AWA is empty when allocated. The user must transfer any desired information from MDBs and PDBs into the AWA. The AWA belongs to the user, and cannot be accessed directly by other users. It is a temporary work space that exists only during a user's session, and is purged at sign-off unless the user takes action to store the information in a Personal Data Base.



8-1. FDS has four major data storage areas.

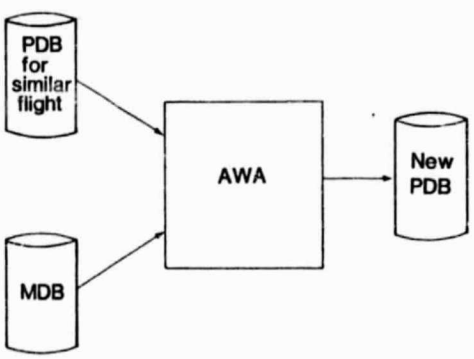
FDS has four major data storage areas: AWA, PBD, MDB and YOD. As a temporary work space allocated at sign-on, the AWA stores data for the processors and is used for constructing and manipulating data and tables. PDBs are used for permanent storage of all or part of the user's AWA. MDBs contain master flight information that is made available to all users. Output data from a Y Processor run are stored in the YOD.

A PERSONAL DATA BASE (PDB) is a disk storage area that the user can create during an FDS session for permanent storage of data and tables. Through the use of the FDS Executive, the user names the PDB and transfers data between the AWA and the PDB. Single files or entire data bases may be transferred in either direction between the PDB and the AWA. As a general rule, each PDB should contain files that are used together. The PDB may be accessed by a single user or a group of users working together. Each user can have multiple PDBs. The user is responsible for data management for the PDBs that the user has created. PDB files are not purged at user sign-off and are deleted only on direct action from the user.

Another major data storage area used by FDS is the MASTER DATA BASE (MDB). MDBs contain data that all users can access. For example, tracking station locations and global constants are maintained on MDBs. FDS users can transfer data from an MDB to their AWA. Users can then modify the AWA copy of this information to fit their own requirements and store it in a PDB. However, users cannot create, modify or write to an MDB. The MDB is a read-only source of data and is created and maintained by FDS support personnel. All MDB names begin with "!!". No other data areas can have names beginning with these two characters.

Lastly, the Y-OUTPUT DATA (YOD) area is where the output from Y Processor jobs is stored. YOD files are created by user-initiated Y Processor runs. Each file name is derived from the job name of the Y Processor run that created it. Files can be transferred from a YOD to the AWA and modified, stored in a PDB, or used in another X or Y Processor run. The user is responsible for managing data in YODs as well as deleting a YOD after it is no longer needed. All or portions of the YOD may be used to create files within the AWA for access by the X Processors.

In preparing to develop the flight design for the example problem, the user might begin by obtaining information about a prior flight which has similar properties. Such data would ordinarily be obtained by accessing the PDB associated with the prior flight. The user might also obtain global constants and similar data from one or more MDBs.



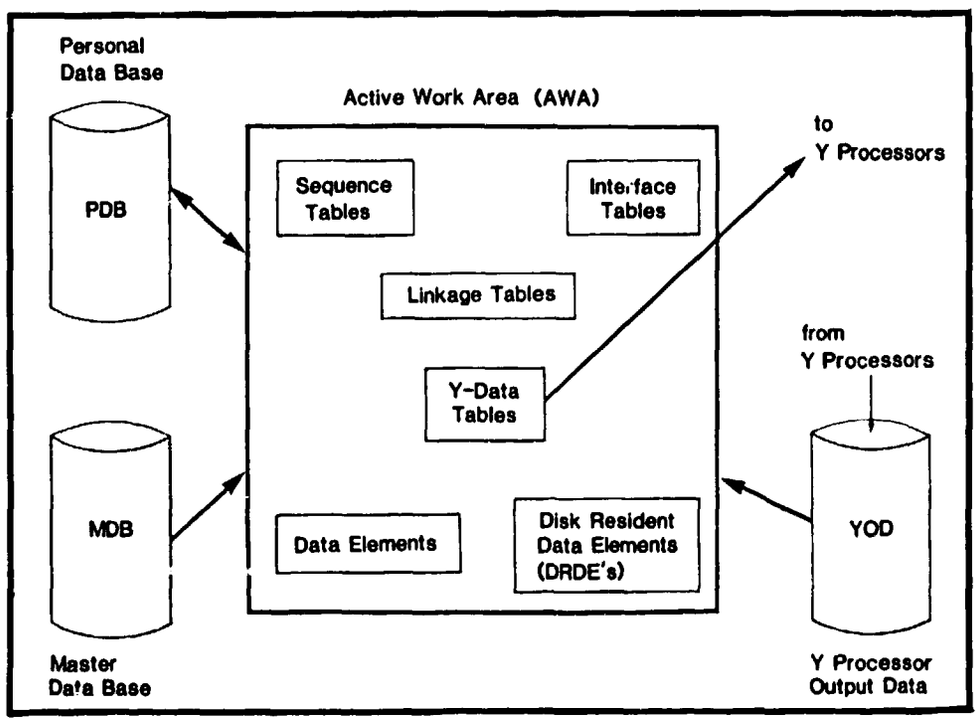
This information is all obtained by moving it into the AWA, where the user can edit it, delete portions of it, etc. When the user is finished with this initial terminal session, the resulting data should be saved by storing it in a new PDB. This PDB can then be accessed in future sessions dealing with this particular flight.

09 FILE TYPES

In addition to the four storage areas, FDS has six explicit file types, which differ in form and purpose.

A DATA ELEMENT (DE) is a collection of related input or output data for a given X Processor. Grouping such information together and storing it under a user-defined name provides a convenient mechanism for managing large amounts of processor input and output data. A DE may be stored in an MDB, in a user's PDB or in the AWA. When it is in the AWA, it can be accessed by processors. Processor-to-processor communication can be achieved by using an output DE from one processor as an input DE for another processor. The user can also create DEs from the information that is returned from Y Processors in the YOD storage area.

Any DE that is too large to be stored in the random access memory portion of the AWA is stored as a DISK-RESIDENT DATA ELEMENT (DRDE). DRDEs are constructed by the execution of processors, which determine their format and contents. Although DRDEs have the same function as DEs, they differ in their form, in the manner in which the data they contain is accessed, and in the fact that users cannot explicitly create a DRDE. A DRDE may be stored in a PDB or MDB, but only the name and characteristics of the DRDE are stored in the core-resident portion of the AWA. Since a DE is structured as an array, the contents can be accessed by subscripting. Because a DRDE has a different structure, its contents must be accessed differently.



9-1. Six types of files can reside in the AWA, PDBs, or MDBs.

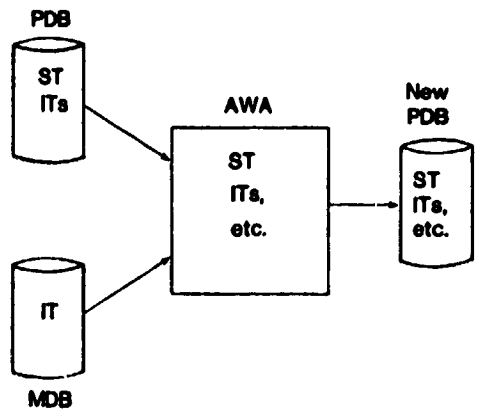
In addition to the four data storage areas already discussed, FDS has six file types. The various file types differ with respect to the user's capabilities to initialize, change, and control their contents. The six file types are DE, DRDE, IT, ST, LT, and YDT. Any of these file types may reside in the AWA, PDB, or MDB storage areas.

An INTERFACE TABLE (IT) is associated with each X Processor and provides a mapping of user-specified input and output data items to the parameters programmed into the processor. X Processors can read data from ITs, and can read and write to and from DEs and DRDEs. For each value read or written by the X Processor, the IT contains either the corresponding value or the name of a DE or DRDE (or, for document production, the name of a linkage table). At the time an X Processor is created, the specifications for its interface table are defined. Each interface table, therefore, is built to the unique specifications of one X Processor. A "default" interface table for each X Processor has been constructed by FDS support personnel. The user may modify this table to create new tables for specific tasks.

A LINKAGE TABLE (LT) is used as input to the documentation system and is identical in structure to an interface table. However, it is associated with a document rather than with an X Processor. It serves to provide a correspondence between the user's data values and the corresponding reserved spaces in a skeletal document.

	Type	Purpose
Data Storage Area	PDB	Long-term storage of user data
	MDB	Long-term storage of master data
	YOD	User-accessible storage of Y Processor output data
File Type	DE	Collection of processor input or output data
	DRDE	Large collection of processor input and output data
	IT	Execution control of X Processors
	LT	Execution control of documentation system
	ST	User-defined and controlled command execution
	YDT	Control of input to Y Processors

9-2. Each storage area and file type has a specific use.



In working on the example problem, information generated about a prior flight may be used. From a PDB, for example, the user could retrieve the sequence table and associated interface tables used on the prior flight. If the example problem involves use of an X Processor that was not used on the prior flight, the interface table associated with this processor could be retrieved from an MDB.

A SEQUENCE TABLE (ST) is a collection of FDS commands stored for later execution. These commands can invoke FDS Executive functions and cause X Processors to be executed. The purpose of a sequence table is to provide the user with the capability for defining and executing often-used sequences of commands. A typical use of a sequence table is to invoke, in turn, each of the X Processors which correspond to the events (e.g., orbital events) on the timeline.

A Y-DATA TABLE (YDT) is a collection of information that defines the control and data inputs required to execute Y Processors. Y-Data Tables provide a means to create and control the input data for Y Processors. They can also refer to DEs used to store output data from X Processors, which can then be used as input data to Y Processors. The Y-INPUT DATA file, which is the actual file that is used by Y Processors is built from the YDT by the FDS Executive.

10 DATA MANAGEMENT

Data management is concerned with creating, storing, accessing, modifying, and deleting files and data bases.

-ST	sequence table
-IT	interface table
-LT	linkage table
-YT	Y-data table
-DE	data element
-DF	disk-resident data element
-PD	personal data base
-MD	master data base
-YJ	Y-output data
-XJ	X Processor job

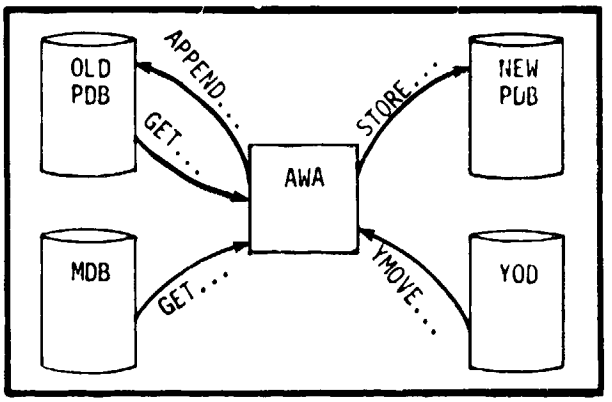
10-1. All file and data base names are unique when they are qualified by a classification code.

A file is a collection of related information; a data base is a collection of files. A group of related information, or data, such as an interface table for a particular processor, is stored as a file. Multiple files, which are also related in some fashion, can be stored as a data base. When qualified by a classification code, all of a user's files and data bases have unique names that are used to access them. Data base names can also be qualified by the access code of the user who creates them. Users can control access to any data bases created under their access codes.

The operations that can be performed on a file depend on the type it is and who owns it. Some files, such as those in a MDB, are "read-only"; that is, they can be read but not modified or written to. Files that are stored in PDBs, however, can be modified.

Files and data bases must be stored on a disk in order to be saved, but some files can reside temporarily in the random-access-memory portion of the AWA. Between FDS sessions, all data bases and files that the user has saved reside on disk. During the FDS session, however, some of these files can be in the AWA. Large files cannot be stored in the random-access memory portion of the AWA and must be kept in the disk-resident portion of the AWA, as DRDEs. The user must be aware of which files are DRDEs, since some FDS functions can only be applied to random-access memory-resident files and not disk files.

Users are responsible for managing any files or data bases that they create or cause to be created. When users create files, they are responsible for saving those files in an appropriate data base, making any modifications to those files, and deleting those files when they are no longer needed. Once created and saved, a file will remain on the FDS until the user explicitly deletes it. Since file storage space is limited, failure to delete old files may prevent the user from having sufficient room to save new files.



10-2. Data can be transferred from one data base or data storage area to another.

X Processors and Y Processors return data to the user; it is the user's responsibility to manage these data. The FDS X and Y Processors aid in the flight design process by calculating data for the user. When using these features, the user specifies where the output data are to go. The user must decide which of this information can be deleted and which must be saved and where to save it.

The FDS user has certain responsibilities for data management. Data management is concerned with creating, storing, accessing, modifying, and deleting files and data bases. Because file types and data storage areas differ in their function, some data management actions apply only to certain files or data storage areas.

When the user creates files and data bases, certain naming conventions must be followed. All files and data bases can be classified as to type and the classification code is sometimes used, in addition to the name, to indicate which file or data base is being referred to. For example, sequence tables are classified with -ST (e.g., TAB1-ST) and interface tables with -IT (e.g., TAB1-IT). All files and data bases that the user creates must have unique names, when they are qualified by type. No two files or data bases of the same type can have the same name.

Different file types exist because they are stored and used differently. For example, Interface Tables and Sequence Tables are used for different functions. Data Elements and Disk-Resident Data Elements are used for similar purposes, but are stored differently and, as a consequence, must be used slightly differently. The user can tell what type a file is by looking at the classification code.

User data can be stored in one of four data storage areas on the FDS. These areas, which were described earlier, are the Active Work Area (AWA), Personal Data Base (PDB), Master Data Base (MDB), and Y-Output Data (YOD) area. Any file type, such as a sequence table, interface table, etc., can be stored in the AWA, PDB, or MDB. A YOD can only contain the data returned from Y Processors. The user can read the information in a YOD, but cannot write to a YOD. Interface tables, data elements, etc., that are to be used by the X Processors cannot be stored on YODs.

One type of executive command allows the user to transfer files among data storage areas. These commands allow the user to save AWA data in one of the other storage areas or to retrieve data from one of these areas into the AWA. The commands that are used differ slightly as a function of the file involved.

ALLOCATE: create DE in AWA
APPEND: append portions of AWA to existing PDB
CLEAR: purge AWA
COPY: copy and rename table, DE, DRDE, PDB, or YOD
DELETE: delete table, DE, DRDE, PDB or YOD
GET: retrieve portions of MDB or PDB into AWA
INSERT: combine portions of tables
RENAME: rename table, DE, DRDE, or PDB
STORE: copy portions of AWA to new PDB
SUBSTITUTE: replace contents of existing PDB
YMOVE: copy portions of YOD to a DE in AWA

10-3. These commands are used to perform data management functions.

In solving the example problem, interface tables, sequence tables, DEs, and DRDEs will be created. All of these files could then be stored in one data base. For example, we could create a PDB for the example problem and call it EXPL-PD.

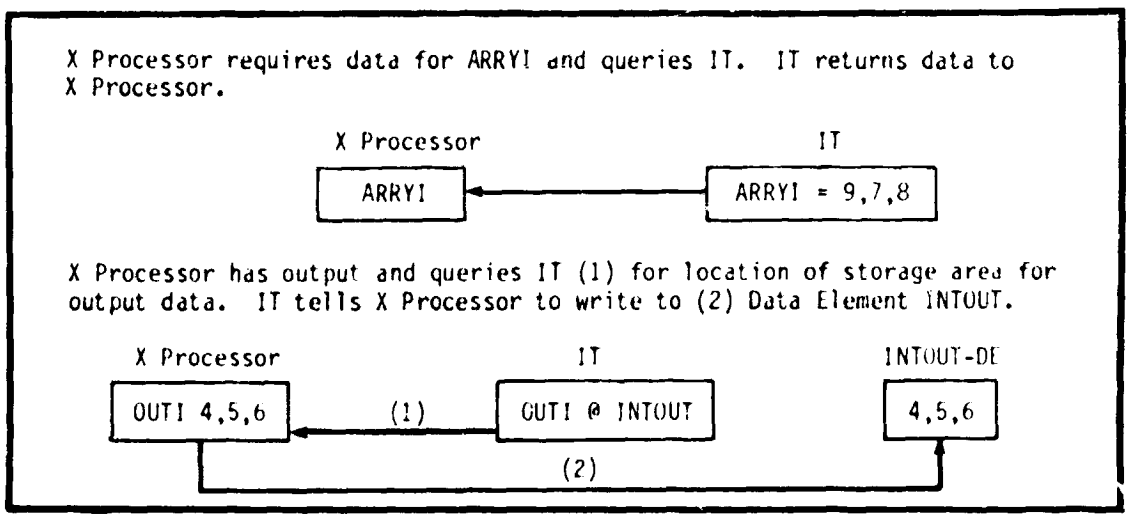
When working on a later flight, it would then be possible to retrieve these files from EXPL-PD, modify them for the current problem, and store them in a new data base. We could also delete the entire data base or selected files, if we were certain they would not be used again.

11 INTERFACE TABLES

Each X Processor has an Interface Table that controls execution of the processor and defines input and output parameters.

An Interface Table (IT) is a named and formatted collection of parameter attributes and values associated with one X Processor. Each IT is constructed to match the specifications of the processor that will use it. At the time an X Processor is created, the specification for its IT is defined and FDS support personnel construct the "default" IT. ITs define a list of input and output parameters for each X Processor, and provide a mapping of the user-specified input and output data to the variables in the X Processor. Since some of the input variables may determine the function of the X Processor, ITs are also used to control the execution of the X Processors. X Processors can read data from ITs, and can read and write to and from DEs and DRDEs. For each value read or written by the X Processor, the IT contains either the corresponding value or the name of the DE or DRDE (or, for document production, the name of a linkage table).

As a processor requires a particular input from an IT, or the name of an output DE or DRDE, the IT is consulted. At the time the IT is consulted, the particular element must be present. If it is not, the IT editor is invoked and the user must supply the missing information. If the missing information is marked as required by the application processor, the editor will be invoked before the processor can be executed. If the information is optional, the editor will only be invoked when the information is needed; that is, during execution of the X Processor. (IT editor functions will be discussed in detail in a subsequent section.) It is important to remember that X Processors can read from ITs, but they do not write to ITs. Input data that are required by a processor can either be read from a DE or DRDE that is named in the IT or be read directly from the IT. Output from processors, however, can only go to DEs or DRDEs whose names appear in the IT; they cannot go directly to the IT.



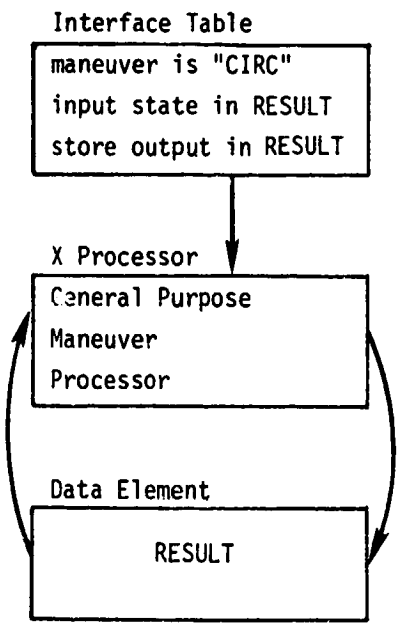
11-1. An X Processor can read values directly from an IT, but can write only to a DE or DRDE named in the IT.

Interface Tables are required for all X Processors. ITs define input and output parameters for X Processors. The format for ITs is pre-defined and can be found in default or existing ITs. Default ITs can be obtained automatically from the system. X Processors can read from ITs, and can read from, or write to, data elements named in the ITs. Linkage Tables are similar to ITs, but are used in the document processing function.

ITs for a particular processor have a constant, pre-defined format. This format is defined by the processor involved. The pre-determined format for ITs can be found in default ITs (which are "templates" constructed by FDS support personnel), or in existing ITs. Each processor has certain required parameters, but may also have optional parameters. The default ITs generally contain default values for those variables which the user is unlikely to change; otherwise, variables in the default IT are incomplete and/or missing. The user can create a specific IT by editing the default IT (or other existing ITs), adding the missing and/or incomplete entries, and saving the new IT. Interface tables generated by the user are automatically stored in the user's AWA; however, the user may also store ITs in PDBs, or automatically receive default interface tables from the system. During processor execution, the IT must be in the AWA. It should be noted that users are responsible for managing any ITs they create. This includes naming, modifying, deleting, etc.

A Linkage Table (LT) is structured exactly the same as an Interface Table, but is used as input to the documentation system. Generally speaking, ITs will point to an LT (as if it were a DE or DRDE) to satisfy data requirements for a particular document segment which is being processed by the document processor. LTs are created in the same manner as an IT, have the same basic structure/format, and changes are made to LTs in the same manner as ITs. The difference is that an LT defines the list of input parameters for a particular document/segment (rather than for a particular X Processor) and is used to control the execution of the document processor. LTs are discussed further in connection with the document processing function.

In the example problem, one of the trajectory maneuvers is the "OMS-2" maneuver used to circularize the initial 150-nm. orbit of the shuttle. The basic computations associated with planning this maneuver will be done by the General-Purpose Maneuver Processor (GPMP). GPMP is an X Processor which computes a single vehicle impulsive maneuver which is initiated at a specified maneuver point in the vehicle orbit and is targeted to a specified final orbit condition. The Interface Table associated with a particular execution of this processor contains information which describes the particular maneuver. In the case of OMS-2, the selected maneuver is identified as "CIRC", for circularization. The Interface Table also names the data elements which are to contain the state vectors which describe the state of the orbiter before and after the maneuver. In the case of OMS-2, this information will be retrieved from, and stored in, a data element called "RESULT". The Interface Table contains a variety of additional information dealing with other selected options, physical units to be used, display(s) to be produced, etc.



12 DETAILED CONTENTS OF INTERFACE TABLES

Each entry of an Interface Table consists of an entry number, a parameter keyword, and information which defines the value(s) of that parameter.

Each entry of an Interface Table consists of an ENTRY NUMBER and a DATA FIELD. The entry number is a numeric field (range 1-170) that uniquely identifies the table entry. Entry numbers are used, for example, to identify entries to be changed. Interface Table entry numbers are preassigned and cannot be changed by the user. The data field in an IT contains parameter names and associated value fields. The value field may consist of one or more data values. Alternatively, it can be a reference to a DE or DRDE where data values may be found for input, or where output values are to be stored. IT variables are of three types -- input, output, and input/output. IT output and input/output variables must contain the names of DEs or DRDEs (in the AWA) where data values are to be found and/or stored. Data values can be stored in an IT only for input variables.

```
 1,GEOCON     =  !!GEOCON                (!! identifies MDB element)
 2,PHYCON     =  !!PHYCON
 3,SESCON     =  !SESCON
 4,DSPCON     =  !SESCON                 (! indicates a configuration controlled
                                          DE built by a processor)
 5,BDATE      =  !BDATE
 6,PROCON     =  10, 1.0E-4, 6.0E-4, 75. (array of literal numbers)
 7,INVEC      =  RESULT(1,3)             (RESULT is a data element allocated by user)
 8,THRUST     =  1.200000E+04
 9,ISP        =  3.131999E+02
10,ENGID      =  "OMS "
11,MANID      =  "CIRC"                  (CIRC is the circularization maneuver)
12,DELVOPT    =  "LVLH"
15,ALTOPT     =  "ALT "
22,CD         =  2.0
23,AREA       =  2690.
24,DENSMODL   =  !!DENSMD
25,SOLAFLUX   =  !!SOLRMD
26,GEOPOTEN   =  &                       (& means missing data to be supplied by user)
27,SVPROP     =  "AEG "
28,DISPLAY    =  "TRM "                  (= symbol describes input data)
29,GPMPTET    @  RESULT(1,3)             (@ symbol describes output data)
30,SUMTAB     @  !CIRTAB
```

12-1. The Interface Table for an OMS-2 trajectory maneuver illustrates many of the possible features of ITs.

An IT entry can be MISSING or INCOMPLETE. An "incomplete" entry is one in which the parameter name is present, but one or more data fields have not been filled in. The & symbol signals FDS of this condition. A given entry may have multiple incomplete fields. The &LABEL symbol is like the & symbol alone, but a label of up to seven characters can be added to aid in identifying what is left incomplete. For example, if the incomplete item is the name of a data element containing a state vector, &STATEV could be used to indicate that this required item is a state vector. If a parameter is not present at all, it is "missing". Since all required parameters are always present (even if incomplete), only optional parameters can be missing.

Values entered in tables may be of different DATA TYPES. Data type refers to the way that the value is represented. For example, integer data types cannot involve decimal points, while real data types can. Each data type (e.g., 6 and 6.0) has a different representation in the computer. Processors require certain data types to be used for each of their input and output values.

ITs consist of entry numbers and associated data fields. Entry numbers identify each table entry. Data fields contain parameter names and value fields. Value fields denote one or more data values, or locations of input data, or locations in which output values are to be stored. & or &LABEL may be used in value fields to indicate missing or incomplete data to be filled in later. Parameter attributes are defined for each X Processor. Attributes displays consist of parameter name, class, type, use, and dimensions.

DATA TYPES

I = INTEGER -- 1, -7, +17
-- consists of digits and an optional sign (+ or -).

R = SINGLE PRECISION REAL -- 6.12, -17.23E01, 12.1E+02, 17.3E-02, 4E2
-- consists of digits, an optional decimal point, an optional sign, and an optional scale specification.

D = DOUBLE PRECISION NUMBERS -- 6.12, -17.23D01, 12.1D+02, 17.3D-02, 4D2
-- like single precision, but uses D instead of E.

Cn = CHARACTER DATA -- "AB", "XX17BC"
-- alphanumeric data enclosed in quote (") marks. Character data items consist of either 4, 8, 16, 32, or 64 characters (e.g., C16). You may use fewer characters than those allowed, but not more.

M = MIXED TYPE -- -7, "AB", 16.2D03
-- a fixed combination of the above types.

F = FREE DATA -- 6, -14.2E03, "EXPL"
-- a random, or unconstrained, combination of the above types.

T = TIME DATA -- 10:9:15:32.1
-- a time expression (see Appendix D).

12-2. An Interface Table may use one or more different data types.

Each X Processor has predefined input/output attributes. The attributes are determined at the time the X Processor is added to the FDS library and can only be altered by FDS support personnel. Attributes of a particular X Processor serve as a foundation from which Interface Tables are constructed. Use of the ATTRIBUTES command will display a given X Processor's attributes. The attributes display provides, for each parameter, its entry number, name, its data class (DE or DRDE), data type, dimensions (size or subscripts), and codes indicating its required or optional nature and its use as input, output, or both.

ATTRIBUTES DISPLAY FOR PROCESSOR GPMP VERSION 6

ENTRY	KEYWORD	SIZE	ROW	COL	CLASS	TYPE	I/O	R/O
1	GEOCON	60	15		DE	D	I	R
2	PHYCON	60	15		DE	D	I	R
3	GSL	4	1		DE	D	I	R
4	LBMKG	4	1		DE	D	I	R
5	FTM	4	1		DE	D	I	R
6	SESCON	288	144		DE	F	I	R
7	BDATE	164	82		DE	F	I	R
8	PROCON	14	7		DE	F	I	R
9	INUEC	56	28		DE	F	I	R
10	THRUST	2	1		DE	R	I	R
11	ISP	2	1		DE	R	I	R
12	ENGID	2	1		DE	C4	I	R

12-3. Use of the ATTRIBUTES command will display the attributes of a specific X Processor.

An Interface Table used by GPMP to circularize the initial 150 nm. shuttle orbit is presented in Figure 12-1. All of the parameter names necessary to calculate the example problem's OMS-2 circularization maneuver are present. With one exception, the individual data fields are complete. Only entry number 26 contains a data field that still must be completed by the user. The IT contains examples of input values (e.g., entry number 6), input DEs (entry number 3) and output DEs (entry number 30), as well as a variety of features discussed in the figure. It would be useful to consider, together, all three of the figures in this module.

13 X PROCESSORS

X Processors are used for flight design computations which do not require detailed simulation, and are fully described in the X Processor User Reference Manual.

As previously explained, one of the major components of the Flight Design System is the library of X Processors. These independent application processors are used as building blocks in flight design. X Processors are designed to perform the computational and documentation support requirements of flight design tasks which lead to CFP development. The fidelity of these processors is consistent with CFP development, but may be insufficiently detailed for OFP development. X Processors may be executed in a multi-user interactive environment or, if the user specifies, X Processors may be executed in a background (batch) mode. In either case, the processors may be executed individually or in sequence.

The General Purpose Maneuver Processor (GPMP) is one X Processor that is frequently used in the flight design process. GPMP computes a single vehicle impulsive maneuver that is initiated at a specified maneuver point in the vehicle orbit and is targeted to a specified final orbit condition. GPMP includes 13 commonly used impulsive maneuvers. However, a particular Shuttle mission may require an impulsive maneuver not present in GPMP singular options. In this case, a combination of GPMP maneuvers may enable the user to achieve the desired results. For a detailed description and explanation of GPMP capabilities and uses, as well as the capabilities and uses of the other X Processors, the X Processor User Reference Manual should be consulted. That manual contains information on the purpose, assumptions, computational methods, and input/output requirements of each X Processor.

GPMP PROCESSOR INTERFACE TABLE

Prompt parameter num/name	Class	Type	Use	Dimension	Values stored in default interface table	Prompt parameter definition
GEOCON	DE		RI		!!GE CON	Geophysical constants array, master data base element
PHYCON	DE		RI		!!PHYCON	Physical and mathematical constants, master data base element
SESCON	DE		RI		!SESCON	Session constants array
DSPCON	DE		RI		!SESCON	Display constants array
BDATE	DE		RI	76	!BDATE	Basetime data array
PROCON	DE		RI	4		GPMP limits and tolerances
INVEC	DE	M	RI	16		Position/velocity input state vector
THRUST	DE	R	RI	1		Vehicle thrust magnitude
ISP	DE	R	RI	1		Vehicle specific impulse, seconds
ENGID	DE	C4	RI	1		Propulsion system identification code (1) "SRB" = Solid Rocket Boosters (2) "SSME" = Shuttle System Main Engines (3) "OMS" = Orbital Maneuvering System (4) "RCSA" = Reaction Control System - AFT (5) "RCSF" = Reaction Control System - FWD (6) "SUS" = Shuttle Upper Stage

N O T E S	CLASS DL DRDL	SIZE F=Freq I=Intg R=Real D=Dubl M=Mixed	C4 C8 C16 C32 C64	USE RI- Required Input RI/RO OI- Optional Input RI/OO RO - Required Output OI/RO OO - Optional Output OI/OO	Dual Usage	Mixed data format specifications: Name(Mtype...Mtype)

13-1. For each X Processor, the X Processor User Reference Manual provides detailed input/output requirements.

X Processors perform computations and provide documentation support for basic flight design and CFP development. They can be executed individually or sequentially, in an interactive or batch mode. The X Processor User Reference Manual contains detailed descriptions of all X Processors.

The following is a list of X Processors and associated ITs used in solving the example problem.

BASTM, T1	--	Establish time reference to be used by all subsequent X Processors.
PHYDM, HOWBIG	--	Establish physical dimensions to be used by all subsequent X Processors.
ASCENT, UP	--	Perform launch computation. Establish state vector at Main Engine Cutoff (MECO).
GPMP, ETSEP	--	Perform computation associated with external tank separation maneuver.
GPMP, OMS1	--	Simulate first OMS burn to achieve parking orbit altitude (150 nm.).
GPMP, OMS2	--	Simulate second OMS burn to circularize parking orbit.
GPMP, SEP1	--	Deploy (separate from) first payload.
GPMP, PKM	--	Simulate upper stage perigee kick maneuver to place first payload at geosynchronous altitude.
GPMP, AKM	--	Apogee kick maneuver to circularize geosynchroncus orbit.
GPMP, TRANS	--	Transition to 200 nm. shuttle orbit.
GPMP, CIRC1	--	Circularize 200 nm. orbit.
GPMP, SEP2	--	Deploy second payload.
ACOAST, ORB	--	Project state vector forward as if "coasting". Forward projection is to allow detailed ephemeris computation.
INVAR, EPHM	--	Compute smooth (detailed) ephemeris.
STCN, GSTDN	--	Determine ground station contact intervals for communication.
SRSS, SUN	--	Compute day/night cycle information (Sunrise, Sunset).
LANDOPT, LAND	--	Compute landing opportunities within crossrange capability of orbiter.
DEORBIT, DEORB	--	For the selected landing opportunity, perform detailed deorbit and landing computation.
GTRACK, GTK	--	Compute ground track for entire mission.
FPD, FLTP	--	Produce Flight Plan Display(s) for entire mission.

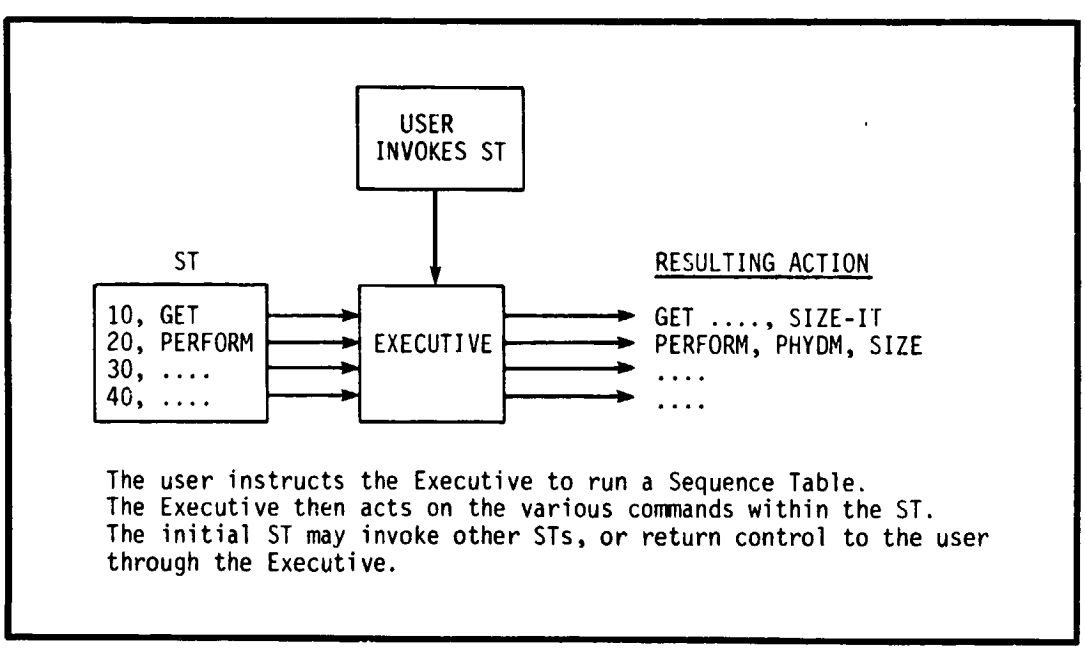
14 SEQUENCE TABLES

A Sequence Table contains a sequence of FDS commands ("statements") which have been stored for later execution.

A Sequence Table (ST) is a named and formatted collection of FDS commands stored for subsequent execution. STs allow the user to define and store often-used sequences of commands. All or part of a Sequence Table may be executed as an entity in either automatic, semi-automatic, or batch modes of execution. Although STs normally contain commands that are executed sequentially, these commands can also be executed repetitively or conditionally.

Sequence Tables may be created, altered, or deleted. STs created by the user are automatically stored in the AWA; however, the user may also store STs in PDBs, or retrieve master Sequence Tables from MDBs. There are no default STs. The user creates new STs "from scratch" or by editing existing STs.

Each entry in an ST consists of an entry number and a command (with its associated parameters). The entry number is an unsigned integer and appears in ascending numerical order (range: 1-32767). All of the Executive commands except OFF, CLEAR and EXIT may be used in Sequence Tables. In particular, DO LOOP, LOOP END, IF, ELSE, and ENDIF may be used only in Sequence Tables. Each command requires a specific order and format for any associated parameters. See later discussions of commands for proper syntax.



14-1. Sequence Tables allow the user to store a series of commands for later execution.

Sequence Tables contain sequences of Executive commands. By storing a series of commands in an ST, the user is able to execute the series simply by invoking the ST. The commands are ordinarily executed in simple serial order, but they can also be invoked repetitively, conditionally, or sequentially. STs are stored in AWAs, PDBs, or retrieved from MDBs. There are no default STs.

The normal approach to solving a basic flight design problem with FDS is to construct a Sequence Table which executes, in turn, each of the required X Processors. The use of a stored "program" allows the user to correct errors and re-execute the computational sequence with minimal effort. It also provides a mechanism for remembering the computational process at any time in the future. The ST for the trajectory portion of the example problem closely follows the list of trajectory events presented previously.

This ST provides the commands and associated data necessary to simulate the example mission through the GROUNDTRACK CHECK event. The majority of the entries in this ST contain a PERFORM statement followed by an X Processor name, and the name of an associated Interface Table. As one might guess, each of these commands causes one of the X Processors to be executed. The NOTE statement is also used frequently (entry numbers 100, 700, etc.). The NOTE statements are not executed by FDS, but are input by the user for clarification purposes. The ALLOCATE statement is also present in the ST (entry numbers 400-600). ALLOCATE allows the user to define certain Data Elements necessary for X Processor calculations. Entry numbers 900 and 1400 contain ASSIGN statements. This statement allows the user to compute values and store them in an existing DE. Finally, the LIST statement is used in the example ST. Entry number 2500 instructs FDS to display the contents of the Data Element called RESULT on the line printer (-P).

SEQUENCE TABLE

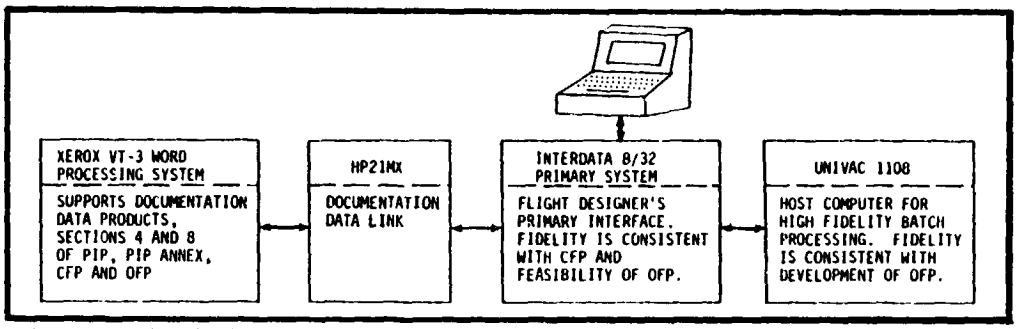
```
 100, NOTE,"INITIALIZATION"
 200, PERFORM,BASTM,T1
 300, PERFORM,PHYDM,HOWBIG
 400, ALLOCATE,INVEC(30,20)-R
 500, ALLOCATE,RESULT(30,20)-R
 600, ALLOCATE,HIALT(30,20)-R
 700, NOTE,"ASCENT"
 800, PERFORM,ASCENT,UP
 900, ASSIGN,RESULT(13,1) = 2.430089E+05
1000, PERFORM,GPMP,ETSEP
1100, PERFORM,GPMP,OMS1
1200, PERFORM,GPMP,OMS2
1300, NOTE "PAYLOAD 1"
1400, ASSIGN,RESULT(13,7) = -1.254199E+04
1500, PERFORM,GPMP,SEP1
1600, NOTE,"UPPERSTAGE PAYLOAD 1"
1700, PERFORM,GPMP,PKM
1800, PERFORM,GPMP,AKM
1900, NOTE,"PAYLOAD 2"
2000, PERFORM,GPMP,TRANS
2100, PERFORM,GPMP,CIRC 1
2200, ASSIGN,RESULT(13,13) = -6.319999E+03
2300, PERFORM,GPMP,SEP 2
2400, NOTE,"DEORBIT CHECK AND DISPLAY"
2500, LIST-P,RESULT
2600, PERFORM,ACOAST,ORB
2700, PERFORM,INVAR,EPHM
2800, PERFORM,STCN,GSTDN
2900, PERFORM,SRSS,SUN
3000, PERFORM,LANDOPT,LAND
3100, PERFORM,DEORBIT,DEORB
3200, PERFORM,GTRACK,GTK
3300, PERFORM,FPD,FLTP
```

15 FDS HARDWARE

The FDS consists of four major hardware components, in addition to several user workstation devices for interacting with FDS.

The INTERDATA 8/32 is the primary system for the FDS. The Executive and X Processors reside on this system. This component is the primary interface between the user and the functions provided by FDS. The Tektronix terminals are directly connected to this component. Through these terminals, the user can access any of the functional components of the FDS.

The XEROX VT-3 Word Processing System supports document production. This component functions as the center for automated documentation. Skeletal document forms reside on the Xerox and can be requested by the user through the Interdata. These skeletal documents are merged with data values and transmitted back to the Xerox for production.



15-1. Each of the FDS hardware components has a unique use.

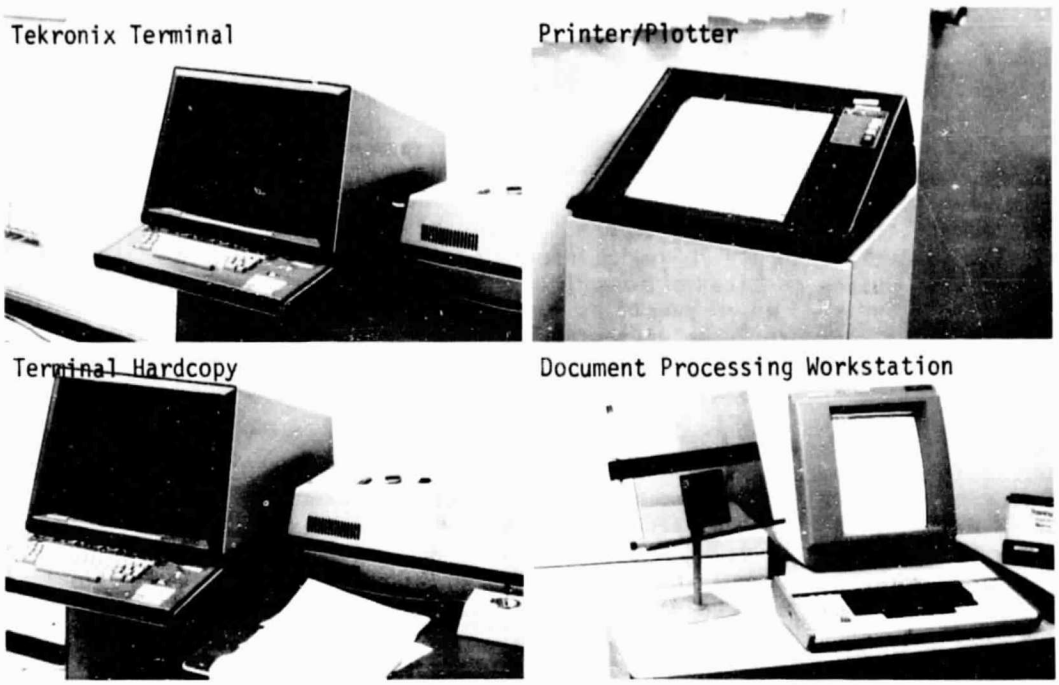
The HP21MX is used as an interface that passes document-related data between the Interdata and the Xerox. Although the HP21MX is a separate computer system and could be used for a variety of purposes, this is the only role it plays as a part of FDS.

The UNIVAC 1108 is the host computer for the Y Processors, which provide the RJE capability of FDS. Y Processors are used to generate high-fidelity OFP data. This hardware component of the FDS is larger than the others and can compute results to a greater accuracy. This component can only be accessed in a remote job entry mode; it cannot be accessed interactively from a terminal, although inputs to and outputs from this component can be done via a terminal. Some outputs from Y Processors, such as figures and graphs, cannot be transmitted back to the user terminal. However, all nongraphical data that are needed for further work with X Processors or for document production can be displayed at the terminal.

Most of the interactions with the FDS are through the terminal. Communications with the FDS are initiated through a TEKTRONIX terminal which is connected to the Interdata. All user commands are initially input through a terminal and most of the output from the FDS is returned to the terminal. There are other devices, however, that also aid in using the FDS.

For high-quality output of graphical data, a printer/plotter is available. The VERSATEK printer/plotter is one device that the user can access. This could be used, for example, to produce ground tracks. There is only one such device and it must be shared by all users.

The FDS consists of four major hardware components -- an Interdata 8/32 primary computer, an HP21MX minicomputer, a Xerox VT-3 Word Processing system, and a Univac 1108. The user workstation provides a variety of devices that allow the user to interact with FDS for a variety of functions.



15-2. FDS has several input/output devices.

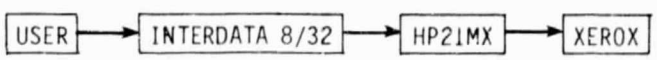
A hardcopy device is associated with each terminal. The user may wish to have a copy of some of the information that is returned to the terminal. This device allows the user to generate a copy of the current contents of the terminal screen, if the user wants to save it.

The document processing workstation uses both a terminal and a special-purpose printer. The terminals associated with the Xerox can only be used for final document production; they cannot be used to access other components of the FDS. Although using this device is the only way in which documents may be printed, preparation for document production can be done through any FDS terminal.

In solving the CFP portion of the example problem, primarily X Processors will be used. This involves use of the Interdata 8/32 primary system.

In order to access Y Processors, for example to solve the OFP portion, the Univac is accessed through the Interdata and data are returned to the Interdata.

In order to produce a document, such as an OFP or a CFP, the user interacts with the Interdata, which creates a communication link with the Xerox via the HP21MX.



16 FDS COMMANDS

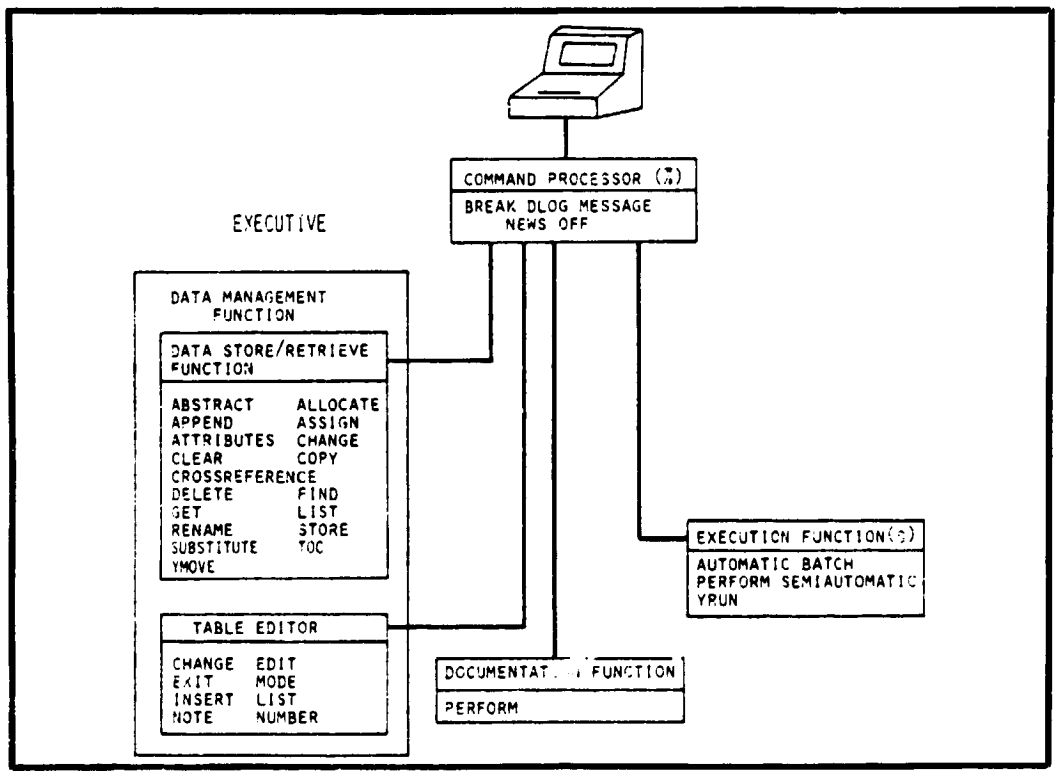
The FDS user communicates with the FDS system through a set of commands.

Each of the operations performed by FDS is supported by FDS commands. The command name entered at the terminal tells the FDS system to do the procedure or procedures indicated by that command. Most command names are followed by a list of parameters, separated by commas. The parameters supply FDS with information about how it is to carry out the command, or what values or files are required. The FDS system interprets each parameter according to a syntax which specifies the order in which the possible parameters are entered. For some commands, some of the parameters must be specified every time the command name is entered. These are called required parameters. Other parameters can be omitted, in which case FDS assumes default values for those parameters.

Commands in the FDS User Guide will always be typed in capital letters. When the syntax of the command is described, the command name will be followed by the set of parameters associated with that command, separated by commas. Required parameters of a command, which must be included each time that command is used, will be indicated by underlines. Optional parameters will not be underlined. Ellipses (...) will be used to indicate that a particular pattern can be repeated at will. An example of the format used for commands in this user guide is shown below.

COMMAND,parameter 1,parameter 2, parameter 3,...

This command takes three parameters, only the first of which is required.



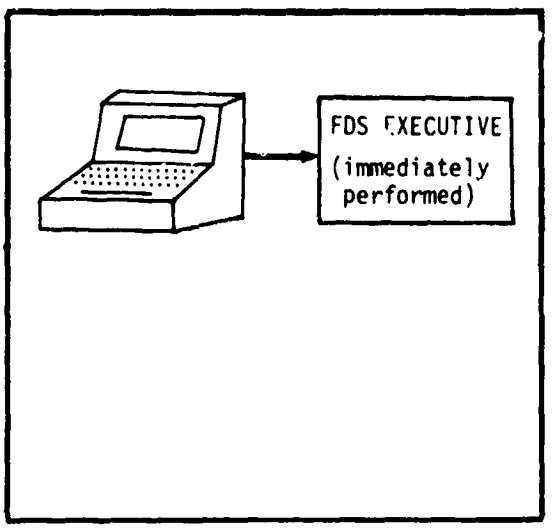
16-1. Each of the operations performed by the FDS Executive is supported by a set of commands.

Each of the operations performed by FDS is supported by one or more commands. The user communicates with FDS by using these commands. The basic form in which FDS commands are stated is

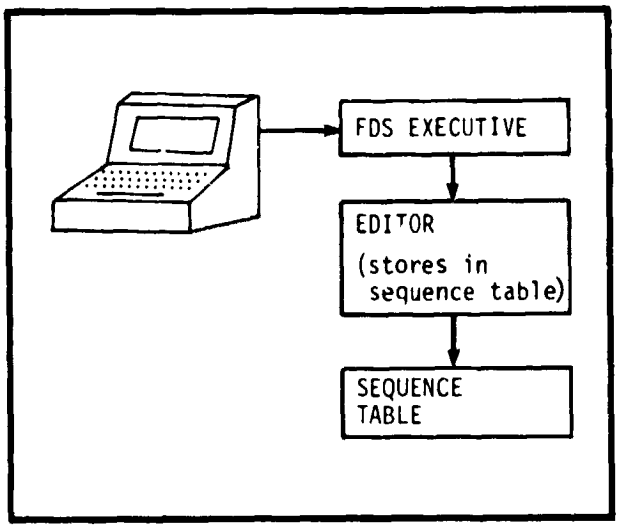
COMMAND,parameter 1,parameter 2,parameter 3,...

When FDS commands are described in the user guide, command names will be shown in capital letters and required parameters will be underlined. A command may be useable as a directive (which is executed immediately), or as a statement (which is stored for later execution), or as either.

Most FDS commands can either be executed immediately or can be stored in a sequence table for later execution. When a command is to be performed as soon as it is entered by the user, it is called a DIRECTIVE. When a command is stored for later use -- much like a statement in a programming language -- it is called a STATEMENT. Statements are stored in Sequence Tables, while directives are executed immediately by the FDS Executive. Although most commands can be used as either directives or statements, there are a few commands which can only be directives, and a few which can be only statements.



16.2. FDS Directives are executed immediately by the Executive.



16-3. FDS Statements are stored in sequence tables for later execution.

To perform the computations associated with the OMS-2 trajectory circularization maneuver, the user uses the PERFORM command. The purpose of the PERFORM command is to cause the execution of a single X Processor. The generic description of the command is

PERFORM,X Processor name,interface table name

For this particular computation, the desired processor is the General Purpose Maneuver Processor, whose abbreviated system name is GPMP. The Interface Table which was constructed for this maneuver was called OMS2. Thus, the actual command will be

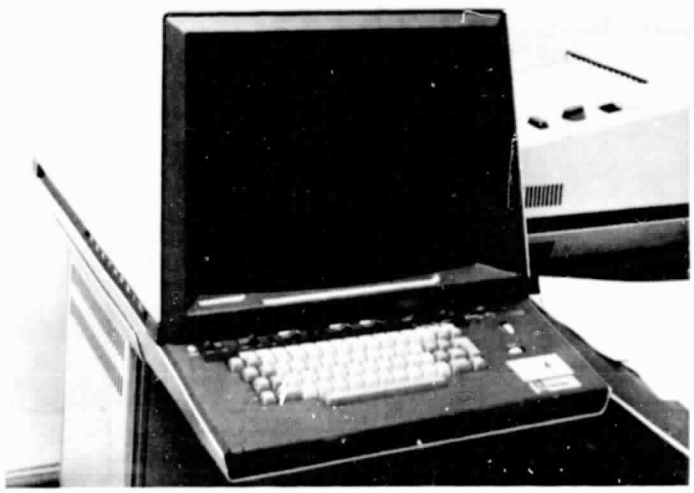
PERFORM,GPMP,OMS2

This command can be stored as a statement in a sequence table, so that it is executed at the appropriate point in a larger sequence of statements. If the user wanted to, though, he or she could type the same command when not editing a sequence table. In that event, it would be a DIRECTIVE and would be executed immediately.

17 USING THE TERMINAL

Successful operation of FDS requires a very basic knowledge of the terminal and its use.

Although a variety of terminal types can be used with FDS, the basic terminal is the Tektronix 4014. This terminal has a very high-quality graphical display capability. Unless the user indicates otherwise when signing on (see later discussion), FDS assumes that this terminal is the type in use. Although the basic FDS dialogue does not make use of graphics, several of the X Processors produce graphical displays if the user is using a Tektronix terminal.



17-1. The Tektronix 4014 is the basic FDS user terminal.

Because the Tektronix 4014 is a storage-tube display terminal, several of its properties differ from ordinary alphanumeric terminals. After the terminal is turned on (the switch is on the pedestal, under the keyboard, on the right), it must be allowed to warm up for a minute or two. The screen must then be cleared (by pressing the RESET/PAGE key at the upper left corner of the main keyboard) before information can be displayed on it. This same key can be used as desired to clear the screen. It is also recommended that the screen be cleared before the terminal is turned off. If no information is input or written to the terminal for a minute or so, the display intensity is automatically reduced in order to prolong the life of the CRT. The display can be reinstated by pressing the SHIFT key. For more detailed information, the user should consult the User Manual for the terminal.

17-2. The Tektronix keyboard has special features because the terminal uses a storage-tube display.

Although many types of user terminals can potentially be used with FDS, the basic FDS terminal is the Tektronix 4014 graphical display terminal. It has some special features which must be understood for satisfactory use, especially as regards display clearing. Other aspects of terminal use which the user should understand include the RETURN key, command abbreviation and termination, and multiple-line commands.

When using any terminal, the user indicates completion of an input by depressing the RETURN key. This key (which is abbreviated "(CR)", for carriage return) must be pressed before FDS can take any action. Thus, when typing in a command, the user must always press RETURN at the end, even though the command definitions do not indicate this. RETURN is not actually a part of the command, but is merely the system's way of recognizing that the user has finished typing.

The user can abbreviate commands by typing in the first two (or more) characters of the command. Thus, the ABSTRACT command can be abbreviated AB, ABS, ABST, etc. Semicolons can also be used to terminate any command, although only a few commands actually require this. Blanks can be typed between arguments in a command, in order to improve readability. Otherwise, commands must be typed as indicated in the individual command descriptions presented in this user guide.

The user should assure the following switch settings:

1. LOCAL/LINE switch set to LINE.
2. ASCII/ALT switch set to ASCII.
3. MARGIN CONTROL set to 1.
4. TTY LOCK depressed.

After turning on the terminal, the user should depress the escape key (ESC), followed by 9 or colon (:), to set the character size. Allowable codes are:

Escape,8 60 characters per line
Escape,9 80 characters per line
Escape,: 120 characters per line
Escape,: 132 characters per line

17-3. For proper terminal operation, the user must assure correct switch settings.

In most cases, FDS allows the user to type a single command on multiple lines. Ordinarily, this facility will be needed only if the command contains many arguments, or a long text string. The system will assume that the user intends to continue typing whenever the user presses RETURN within a literal character string

```
ABSTRACT,MYPDS,"THIS IS (CR)
CONTINUE:MY ABSTRACT"(CR)
```

or immediately after a comma

```
ABSTRACT,MYPDS,(CR)
CONTINUE:"THIS IS MY ABSTRACT"(CR)
```

Notice that the system automatically prompts the user to continue by displaying

```
CONTINUE:
```

Such commands can be continued over several lines, as necessary.

If the user wishes to indicate a line break within a text string, Control-C is used. Control-C is abbreviated ^ in this user guide. This is the character that is generated when the user presses the key for the letter C while holding down the "CONTROL" key. This action, within a character string, causes the system to start a new line whenever this part of the text string is reached. Thus, the input

```
ABSTRACT,MYPDS,"THIS IS^MY ABSTRACT"(CR)
```

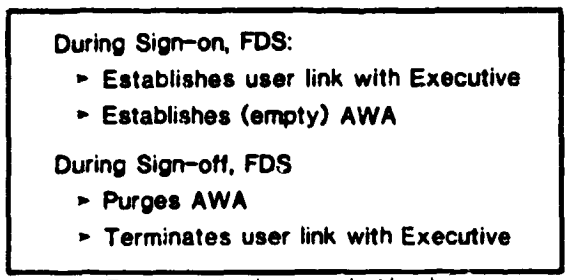
causes the new abstract to be

```
THIS IS
MY ABSTRACT
```

18 Signing On and the OFF Command

When the user signs on or off of the FDS, certain events occur automatically.

Two events happen both when the user signs on to the FDS and when the user signs off. At sign-on, communication is established with the FDS Executive and an AWA is assigned to the user. At sign-off, communication with the Executive is terminated and the AWA is purged.



18-1. Two events happen both when a user signs on and signs off of FDS.

Only qualified users may sign on to the FDS. Each user of the FDS is assigned a two-character identification code, called an ACCESS CODE. This code is assigned by FDS support personnel and is used to sign on to the FDS. The user may sign on through any terminal that is displaying the FDS sign-on prompt "FDS Sign-on:". The access code is also used to identify the owner of a PDB. In order to access another user's PDBs, the corresponding access code must be known.

```
access code,data base name-class-access code,sequence table name,terminal type,DWA size
```

18-2. The user signs on with an access code rather than with a command.

The user can cause a sequence table to be automatically executed at sign-on. At sign-on, the user can name a sequence table and it will be executed automatically. This sequence table could, for example, cause desired files and data bases to be retrieved into the AWA, system messages to be printed, etc. The user specifies the ST by giving both the name of the data base (e.g., PDB) in which it resides and the name of the ST itself.

FDS Sign-on:UC	sign-on under access code "UC"
FDS Sign-on:UC,INIT-PD-OC,TAB1	sign-on and cause sequence table TAB1, stored in PDB INIT and belonging user "OC" to be executed
FDS Sign-on:UC,SET-MD,TAB3,,64	sign-on and cause TAB3 in MDB SET to be executed and set DWA size to 64 sectors
FDS Sign-on:UC,,,,64	sign-on under access code UC and set DWA size to 64 sectors

18-3. There are several ways in which to sign on.

At sign-on, the user may also specify the terminal type and the desired size of the disk portion of the AWA. If no terminal type is specified, the Tektronix 4014 is assumed, and graphical displays may be presented on the terminal. The final argument specifies the maximum number of sectors to be allocated for the disk-resident portion of the AWA (see below). This field is an integer in the range of 1-8192. The specified number will be rounded up to the nearest 16-sector block. If this argument is omitted, 400 sectors (25 16-sector blocks) is assumed.

To sign on:
access code,data base name-class-access code,sequence table name,terminal type,DWA size
To sign off:
OFF
In order to use the FDS, users must first sign on. Signing on establishes communications with the FDS Executive and allocates an AWA for each user. These effects are reversed at sign-off. Only qualified users, who have been assigned an access code, may sign on to the FDS.

The disk portion of the AWA is called the DISK WORK AREA (DWA). The DWA is an extension of the AWA. The AWA has a fixed size. Any overflows of data from the AWA are accommodated in the DWA. Except for DRDEs, which are always stored in the DWA, the user need not be concerned with whether a table, file, etc.&is in the random access memory portion of the AWA or in the DWA; the FDS Executive automatically retrieves DWA data to the memory portion of the AWA when it is needed. All DWA management functions and data transfers are accomplished automatically by the FDS Executive. At sign-on, the user need only specify the desired size of the DWA, if different from the default value.

```
Enter the directive OFF in response to Executive's % prompt
    %:OFF
Then FDS will issue a sign-off warning prompt:
    *WARNING*EX*XDOFF*NNN*
    SIGN-OFF REQUESTED FOR USER ID, AWA TO BE PURGED
    %RESPOND "YES" TO PURGE AWA SIGN-OFF, "NO" TO ABORT SIGN-OFF: (response)
```

18-4. Signing off is done with the OFF command

Signing off is accomplished with the OFF directive. When the user enters this directive, FDS will issue a prompt at the user's terminal, warning that the user's AWA will be purged unless action is taken to save information in the AWA. Any action to save AWA data must be taken before the sign-off is completed. If data are to be retained, and have not previously been saved, the user may abort the sign-off procedure by responding "NO" to the prompt. This will terminate the sign-off procedure and re-establish communication with the Executive. A response of "YES" to the prompt will cause the sign-off procedure to be completed.

When working on the example problem, assume that the information that has been generated so far is stored in a PDB called EXPL1. A sequence table, for example INIT, could be constructed to cause EXPL1 to be loaded into the AWA. Assuming that INIT is stored in a PDB called SET, which belongs to user UC, INIT could be executed automatically by signing on in the following way:

```
UC,SET-PD-UC,INIT,,800
```

This particular sign-on entry also requested that the DWA size be set to 800 sectors.

INIT-ST

```
100, GET,EXPL1-PD
200, TOC
  .
  .
  .
```

19 SYSTEM MESSAGES

Certain conditions will cause FDS to generate a system message that may, or may not, require a user response.

A system message, initiated and generated by FDS, informs the user of some condition or occurrence within the Executive or a processor. A system message will include a severity code, origin of the message, name of the program issuing the message, a number identifying the displayed message, and message text explaining the situation. The severity code will consist of either ERROR, which indicates that a failure occurred, terminating the immediate execution process, WARNING, which informs the user that a problem nas been detected which may require action (at the user's discretion), or INFO, a system message that contains instructional or guidance information. The origin of these messages will either be the Executive or a particular processor. As a general rule, the user is not required to respond to these types of messages. The user ordinarily will not make use of the program name and message number, but may use this information in requesting a more detailed explanation of the error.

By typing in the identifier of an error message, followed by a question mark, the user can obtain a more detailed explanation. If, for example, the message is identified with the characters (*ERROR*PR*GPMP*001*), the user can obtain more detailed information by typing the characters (*ERROR*PR*GPMP*001*?).

```
*ERROR*PR*GPMP*001*
INPUT VECTOR IS NOT TEG/CARTESIAN
%:*PR*GPMP*001*?
*** MESSAGE PR*GPMP001
INPUT VECTOR IS NOT TEG/CARTESIAN
*** TUTORIAL
MEANING: THE USER HAS INPUT A VECTOR THAT IS NOT IN THE TRUE EQUATOR AND
GREENWICH MERIDIAN OF EPOCH CARTESIAN COORDINATE SYSTEM.
ACTION REQUIRED: TRANSFORM THE VECTOR TO TEG/CARTESIAN
```

19-1. After a system message, the user can request more information.

If there is an abnormal processor termination (abend), the user will be notified by FDS. In most cases, an abnormal processor termination is caused by an error in the input data. The user is notified of the processor abend and recovery procedures will depend on the mode of operation. The PERFORM command which caused the processor to be executed may have been entered directly by the user, or it may be in a Sequence Table. In all cases, the PERFORM command is terminated. If a ST is being executed in the AUTOMATIC mode, execution of the entire ST is also terminated. While in the SEMIAUTOMATIC mode, an error will cause the user to be reprompted with the same entry of the Sequence Table which caused the abend to occur. In the BATCH mode, an error will terminate the job and the user is notified of the abend via the BATCH printed output. In any case, output data may be nonexistent, or only partially available, depending on where the abend occurred. The user is responsible for examining the data and performing the necessary housekeeping functions to correct the error.

If the user initiates a processor abort (abend), FDS will generate a message requesting next action. When there is no outstanding prompt on the user's terminal, the user can strike the break key to obtain the following message from FDS.

```
*INFO*EX*XMNTR*nnn*
USER INITIATED INTERRUPT
ENTER REQUEST - STATUS SUMMARY (SS), ABORT (%), OR CONTINUE (CR):
```

FDS generates system messages that explain certain conditions. ERROR, WARNING, or INFO messages describe conditions within the Executive or a processor. Mode of operation will determine FDS notification procedures during a processor abnormal termination (abend). User-initiated processor abend (using the "break" key) will result in a status summary display, abort of processor operation, or continuation of processor execution, at the option of the user.

MODE	DESCRIPTION
Manual	User executes one processor by typing PERFORM command.
Semiautomatic	System executes Sequence Table, one statement at a time, as a result of SEMIAUTOMATIC command. User is given opportunity to alter or veto each statement before it is executed.
Automatic	System executes entire Sequence Table as a result of AUTOMATIC command. There is user interaction only if the processor requires information not provided by the Interface Table.
Batch	A Sequence Table is submitted, via the BATCH command, for later execution by the system. There is no interaction with the user, who may not even be signed on at batch execution time.

19-2. Mode of operation will affect error recovery.

If the user responds with (SS), FDS will display the functional status of all active tasks associated with the terminal. If a (CR) (carriage return) is struck in response to the user initiated interrupt, no action is taken and the processor continues to execute. When the (%) symbol is used, a second prompt is issued to solicit concurrence to abort and to display information about the program name, stack position, and monitor identification. The prompt is: (task name) TO BE CANCELLED, REPLY YES TO CANCEL, NO TO ABORT CANCEL: A YES response will abnormally terminate the currently active program. The user is notified of the program termination and reprompted appropriately. A NO response causes the processor to continue to execute.

When there is an outstanding prompt on the user's terminal, and the user wishes to abend the executing command or processor, the user merely enters the (%) symbol. At that point, FDS notifies the user of the processor termination and the user is reprompted appropriately.

BREAK KEY STATUS DISPLAY

SYMB	PROGNAME	OLDTAB-CLASS		NEWTAB	SEQ #
#	XDEDIT		- ST	Z1	000000
\	XDEDIT		- IT	Z1	000000
$	XDXCTL	T1	- ST		000010
$	XDALLO		-		000000

19-3. If (SS) is keyed during a user initiated interrupt, FDS will display a status summary.

20 EDIT Command, EXIT Command

The editor is used to construct new tables and alter old tables.

The editor is that portion of the FDS Executive that provides the user with the capabilities to create new tables or modify existing tables. The user can edit interface, sequence, linkage, and Y-data tables.

With the editor, the user can:

- ► create a new sequence or Y-data table
- ► create a copy of a default linkage or interface table and modify it
- ► create a copy of any existing table and modify it
- ► modify an existing table
- ► add new entries to any sequence or Y-data table
- ► complete entries with incomplete or missing data fields
- ► modify data fields within an entry

20-1. The editor is used to create and/or modify tables.

All editing is performed on a temporary copy of a table. The FDS editor creates a temporary, duplicate copy of a table being edited. Changes are made only to this temporary table. If the editing function is aborted, no changes are made to the table being edited. If the editor is terminated normally, then the modified table is stored in the AWA, where it can remain for the duration of the terminal session. In order to make these changes permanent, the revised table must be copied to a PDB.

The editing function can be invoked in either of two ways -- explicitly or implicitly. The editor is invoked EXPLICITLY by use of the EDIT command. The user does this in order to create or modify a table and store it for later use.

The editor can be IMPLICITLY invoked when the system is attempting to use a table that has incomplete or missing items. In this case, the system automatically invokes the editor in order to allow the user to provide the required information. When this occurs, the editor prompts the user to provide the specific information that is missing. A typical implicit editor invocation is the situation in which an X Processor is executed, and the interface table has missing or incomplete data. The editor is invoked because the data must be provided before the processor can continue. In the case of the interface table, the user provides the missing data, and this information is relayed to the processor, but it is not stored in the interface table. If the same table is used again, the same missing or incomplete values must be provided.

Because tables differ in their content, a slightly different editing capability is provided for each type of table. The FDS editor is divided into four areas, each with its own unique prompt symbol. Each area corresponds to a type of table and the editing operations that can be applied are a function of the type of table involved.

#	-	sequence table editing function
\	-	interface table editing function
<	-	linkage table editing function
>	-	Y-data table editing function

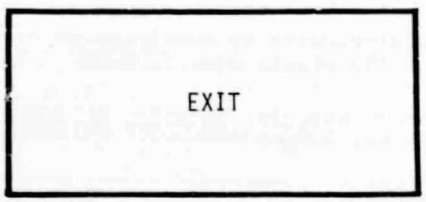
20-2. There are four functional areas of the table editor.

The EDIT command provides access to the FDS capabilities to create and modify tables. This command invokes a table editor, whose behavior differs depending on the type of file (classification code) being edited.

EDIT,current table name, new table name-classification code,reference name

The EXIT command causes a normal termination of an editing operation, including storage of the modified file. An editing operation can be aborted by typing the % character.

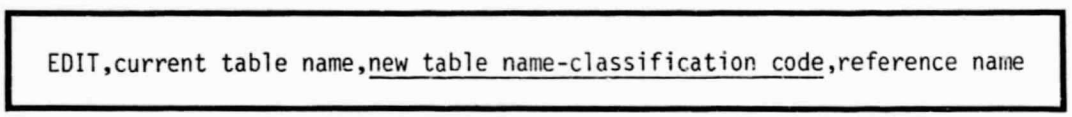
EXIT



20-3. The syntax of the EXIT command.

Normally, the editor is terminated by typing in the EXIT command. EXIT is a directive only, and cannot meaningfully be stored in a sequence table. When the user types EXIT, the modified table is stored in the AWA (if the editor was implicitly invoked). Of course, the table must be copied to a PDB if the user wishes to save it beyond the current terminal session.

The editor can also be aborted, without saving any results of the editing operation. The user does this by typing the % character. This character is the normal prompt symbol for the Executive. By typing it while in the editor, the user is asking for an immediate return to the condition which existed before the editor was invoked. No information is saved from the editing operation, and the table being edited remains unchanged. The user can do this even if the editor was implicitly invoked.



20-4. The syntax of the EDIT command.

The EDIT command must name a "new table" and may also name a "current table" and a "reference name". The NEW TABLE name specifies where the table being edited will be stored when the editor is EXITed. The CURRENT TABLE name indicates the table to which modifications will be made. The REFERENCE NAME is used to indicate the X Processor for which a new interface table is being constructed or the document/segment associated with a linkage table. If the current name is omitted, then a new table will be created. If both the current table name and new table name are specified, then changes made to the contents of the current table will be stored under the new table name. If the current table name and new table name are the same, then modifications are actually made to the current table.

Command	Effect
EDIT,,NEW-ST (create)	EDIT ⇒ NEW TABLE
EDIT,,NEW-IT,PROCA (create)	DEFAULT IT ⇒ EDIT ⇒ NEW IT
EDIT,OLD,NEW-IT (copy and modify)	OLD TABLE ⇒ EDIT ⇒ NEW TABLE
EDIT,TAB1,TAB1-ST (modify)	CURRENT TABLE ⇒ EDIT ⇒ MODIFIED CURRENT TABLE

20-5. There are four ways to use the EDIT command.

21 Using the Editor, MODE Command

The editor interacts with the user in one of several modes, depending upon the type of table being edited and what the user wants to do with that table.

The FDS Editor can operate in any of five modes. The CREATE mode is used to create a new sequence or Y-data table. These are the only types of tables that the user can create directly. Interface and linkage tables can be created only by modifying an existing table. This is done by using the UPDATE mode. The update mode is used to insert new table entries and modify existing table entries. It is the only editing mode in which directives to the FDS Executive can be given. For example, deleting an entry, with the DELETE command, can only be done in this mode. The INCOMPLETE mode causes the editor to read through the file, looking only for instances of incomplete data. Each of these incomplete entries is displayed, and the user is prompted to provide the needed information. The user can provide the data or skip to the next parameter for which no value is present in the table. The MISSING mode behaves the same way, but cycles through all instances of optional data which are missing from the table. The missing mode can be used only with interface tables and linkage tables, since these are the only table types which can have "optional" parameters. The ALL mode starts at the top of a table and displays each entry in turn. In addition to a simple display of every line, the user is prompted and allowed to edit any entries that are either incomplete or missing.

MODE	PURPOSE	Can be Used With			
		IT	LT	ST	YDT
CREATE	Create new table	No	No	Yes	Yes
UPDATE	Modify table entries (Executive directives allowed)	Yes	Yes	Yes	Yes
INCOMPLETE	Find and edit parameters which are incomplete	Yes	Yes	Yes	Yes
MISSING	Find and edit optional parameters which are missing	Yes	Yes	No	No
ALL	Scan and edit entire table, one entry at a time	Yes	Yes	Yes	Yes

21-1. The editor operates in any of several modes.

The mode required to create a new table depends on the table type. A new sequence or Y-data table can be created either by editing an existing table or actually creating a new table. For these two types of tables, therefore, the user may either initialize the editor with an existing table or directly enter the create mode. Interface and linkage tables can only be created by editing an existing table or the corresponding default table.

The FDS Editor will be initialized in one of several modes, depending on the type of table being edited and the status of that table. Each mode is used to perform different editing tasks. The MODE command allows the user to select a desired mode.

The manner in which table entries are referenced is determined by the form of the table entries. Sequence and Y-data table entries consist of commands ("statements") and character strings, respectively. For each entry, the user must supply an entry number, and individual entries are referenced, or accessed, by this number. Although entries in interface and linkage tables are also numbered, the user cannot alter these numbers. Entries in these tables consist of parameter keywords and parameter values and they are referenced by the parameter keywords.

#	Sequence table
\	Interface table
<	Linkage Table
>	Y-Data table

21-2. The user types a prompt character to switch to the update mode.

Depending on the state of the table being edited, the editor will initially be invoked in one of three modes. If a table has no incomplete or missing entries, then the update mode will be used. If a table has incomplete entries, then the incomplete mode will be used. The create mode will be invoked only if the user is creating a new sequence or Y-data table.

The MODE command allows the user to select an editing mode. When an editor is explicitly invoked, it will be in either the update, incomplete, or create mode. The update mode is, in a sense, the basic mode of the editor. It is, for example, the only mode in which FDS Executive directives can be typed in by the user, for immediate execution. When the editor is not in the update mode, the user can switch to the update mode by typing the prompt character associated with the particular type of table being edited. Once in the update mode, the user can, if desired, select another mode by using the MODE command.

```
MODE,descriptor

A  --  All          MODE,A
C  --  Create       MODE,C
I  --  Incomplete   MODE,I
M  --  Missing      MODE,M
```

21-3. After the editor has been invoked, the MODE command can be used to select a desired mode.

In a previous module ("DETAILED CONTENTS OF INTERFACE TABLES") a sample interface table for the GPMP processor was presented. That IT was called OMS2, and was incomplete. Thus, the command

```
EDIT, OMS2, OMS2-IT
```

causes the editor to be invoked in the INCOMPLETE mode, and the editor generates the prompt

```
\260,GEOPOTEN=:
```

This prompt asks the user to provide a value for the parameter GEOPOTEN. Ordinarily, the value will be the name of a data element which defines the mathematical properties of the geopotential model to be used by GPMP.

Once the user types in this value, the IT is complete, since all other required values have already been provided. The editor then reverts automatically to the UPDATE mode, presenting the prompt

```
C\:
```

The user can now type EXIT, or do any other desired editing operations.

22 BASIC EDITOR DIRECTIVES AND RESPONSES

Within the editor, there are several responses that can be used to obtain help or to perform common operations, regardless of the type of table being edited.

Each of the editor modes, except for the "all" mode, has a unique syntax, and different user responses are allowed. When an editor is ready for a user response, certain information is displayed to the user. This information differs as a function of mode, as do the responses that will be accepted. The "all" mode uses the syntax of the "update" mode if an entry is complete and that of the "incomplete" mode if an entry has incomplete fields.

RESPONSE	OPERATION
carriage return	skip to "next" entry
%	abort and return to previous condition
?	display list of statements, directives, or parameter keywords
directive	execute named FDS Executive directive
directive?	display syntax of named FDS Executive directive
&	incomplete item indicator
&LABEL	labeled incomplete item indicator

22-1. Some user responses are common to the editing of all table types.

A CARRIAGE RETURN, used by itself, is a directive that causes an editor to skip to the next entry. The "next entry", however, depends on the mode that the editor is currently in. In the "all" mode, a carriage return causes a skip to the next entry. In the "missing" and "incomplete" modes, a carriage return causes the editor to display the next missing entry or the next entry with incomplete data fields. In the "update" and "create" modes, a carriage return has no effect.

Certain special symbols can be input by the user to abort the editor, or to return to the update mode. The user can always abort the editor and return to the prior condition by entering the % symbol. To get into the update mode, the user types the special symbol corresonding to the type of table being edited.

The ? symbol is used to obtain information about the responses which the user can "legally" make in the current situation. If the editor is in the incomplete mode or the missing mode, the user has been prompted with a request for specific information (e.g., the value which is to be associated with a keyword. In this situation, the ? can be used to obtain an "extended prompt" which provides additional information about the incomplete or missing item (e.g., the definition of the interface table parameter whose value is missing). On the other hand, a ? in the update mode causes the system to list all FDS directives and, in some cases, additional helpful information.

Although the FDS Editor has somewhat different behavior for each type of table, some user responses can be used regardless of the type of table being edited. These responses can be used to obtain help or to perform common operations.

An FDS command that is to be acted on immediately is a "directive". Such directives can only be used in the update mode (or, of course, outside the editor altogether). If the directive that is named is not valid, no action is taken. Otherwise, the requested directive is executed. The syntax of any valid directive can be displayed by typing the name of the directive, followed by a question mark.

	update	create (ST, VDT)	incomplete	missing (LT, IT)	all
carriage return		X	X	X	
%	X	X	X	X	X
?	X	X	X	X	X
revert to update		X	X	X	X
directive	X				
directive?	X				
&		X	X	X	X
&LABEL		X	X	X	X
LIST	X				
DELETE	X				
EXIT	X				

22-2. Editor responses can only be used in certain modes.

The & specifies that a data item is to be left incomplete. The & can be typed as part of an original entry, or can be given as a response to a prompt in the incomplete, missing, or all modes. The ampersand is actually stored as part of the table, and is the signal to FDS that this entry is incomplete. A given entry may have multiple incomplete values. In this case, the editor automatically assigns a sequence number to the incomplete item for display purposes. For example, an input of:

```
10,LIST,&,TAB3-ST,&,TAB4-IT
```

would be displayed with unique sequence numbers after the ampersands, as:

```
10,LIST,&1,TAB3-ST,&2,TAB4-IT
```

The &LABEL "placeholder" is like the &, but a label of up to 7 alphanumeric characters has been added to aid the user in identifying what is left incomplete. When the label is used, incomplete items are not numbered by the editor.

```
LIST,(first entry number - last entry number)

DELETE,(first entry number - last entry number)

Examples:
          LIST,(10-40)
          DELETE,(30-30)
```

22-3. When used as directives in the update mode, the LIST and DELETE commands can be used in simplified form.

The LIST and DELETE directives can be used within an editor. These directives can be used to list or delete all or part of the table being edited. These functions can also be used as FDS Executive commands and the syntax that is used in the editor differs from that used outside the editor. Within the editor, the user specifies the range of lines to be listed or deleted. Typing only LIST, with no parameters, will cause the entire table to be listed.

23 INTERFACE TABLE EDITING

The nature of interface tables determines some of the actions the user can take during IT editing.

The structure of an interface table is determined by the input/output requirements of the X Processor which will use it. The IT structure is determined by FDS support personnel, and cannot be changed by the user. Thus, keywords and entry numbers are fixed. The only actions available to the user are those associated with adding, deleting, or changing values associated with the keywords.

To insure that all tables have the correct structure, new tables can be created only by editing existing tables. Thus, the create mode does not exist for IT editing. The user can create a new table by modifying an exising one, changing only those values which are not already correct. Alternatively, the user can start with the default table (by providing no "existing table name" in the EDIT command). In this case, all values not present in the default table must be provided by the user.

MODE	USE
UPDATE	edit, list, change, etc., with Executive directives
INCOMPLETE	view and change incomplete entries
ALL	view each entry from start of table
MISSING	view and change missing entries

23-1. Interface table editing can be done in any of four modes.

The way in which FDS prompts the user depends on the mode and the state of the IT entry. When the editor is initially invoked, it starts in the INCOMPLETE mode, giving the user a chance to complete any incomplete entries. (Of course, if there are no incomplete entries, the editor automatically switches to the update mode.) Each incomplete entry generates a prompt which shows the entire incomplete entry and gives the user an opportunity to specify one or more missing values. The user is prompted for each succeeding nonexistent value of the entry, until all such values have been addressed:

```
10,A=1,&ABLE,&1,4
\&ABLE:2
\&1:3
```

In this example, the user has provided the values 2 and 3, respectively, for the nonexistent values. A similar prompt is provided in the ALL mode whenever an incomplete entry is encountered. A slightly abbreviated prompt is provided in the MISSING mode, since it is known that no value field yet exists. This prompt shows the entry number, parameter name, and the equivalence symbol which shows whether the entry is an input (=), output (@), or input/output (=@) parameter:

```
40,MODEL=:!!MODNAME
```

In this example, the user has typed the value !!MODNAME after the prompt. In the UPDATE mode, the prompt consists simply of the IT editor symbol:

```
\:
```

The interface table editing function is used to create new interface tables from default or other existing tables, and to modify existing interface tables.

The user may respond to IT editor prompts with a subscript and value. For example, if the parameter INVALS is an array, the user can enter values into its third and fourth elements by responding

```
40,INVALS=:(3)7.,8.
```

This input causes INVALS(3) to have the value 7.0, and INVALS(4) to have the value 8.0.

For cases in which the user must provide a list of values which has significant repetition, FDS provides a "shorthand" method of inputting the values. The REPEAT LIST allows the value or group of values to be typed only once, with a repetition factor which indicates how many times it is to be repeated. Thus, the input 20R5 is equivalent to typing twenty separate values, each consisting of the number five. If more than one value appears in the repeat list, the list must be enclosed in parentheses, as in 20R(1,2,3). These expressions can even be nested to a depth of four or less, as in 20R(1,2R5). No subscripts may appear within a repeat list. A repeat list may contain only literal values or incomplete indicators (&). If a table entry requires or is provided with reference to a DE, DRDE, or Linkage Table, only one such reference is needed and a repeat list cannot be used.

"REPEAT LIST" FORM	EQUIVALENT
4R5	5,5,5,5
1,2R2,3R3	1,2,2,3,3,3
3R(5,7,"AB")	5,7,"AB",5,7,"AB",5,7,"AB"
2R(2R3,"XY")	3,3,"XY",3,3,"XY"

23-2. The repeat list can be used to repeat a list a values.

24 AN EXAMPLE OF INTERFACE TABLE EDITING

The preparation of the interface table for the OMS-2 circularization maneuver provides a good, fairly typical example of IT editing.

A previous module, "DETAILED CONTENTS OF INTERFACE TABLES", presented the OMS-2 IT as an example. This module will discuss the use of the editor to construct that IT. Assuming that the user starts with the default interface table for GPMP, rather than an existing IT, the command which starts the editing process is

```
$:EDIT,,OMS2-IT,GPMP
```

Because there are required parameters, for this interface table, which do not have default values, the table is incomplete. Therefore, the editor starts in the incomplete mode, giving the user a chance to provide values for these parameters.

```
60,PROCON=&1,&2,&3,&4
\&1:10
\&2:1.0E-4
\&3:6.0/E-4
\&4:75.
```

In the case of PROCON, the user provides four GPMP tolerance values to be used in the computation. Similarly, the user provides values for other required parameters

```
70,INVEC=&1
\&1:RESULT(1,3)
80,THRUST=&1
\&1:1.2E+04
90,ISP=&1
\&1:3.131999E+02
100,ENGID=&1
\&1:"OMS"
110,MANID=&1
\&1:"CIRC"
270,SVPROP=&1
\&1:
```

Whenever the user needs additional information about the parameter, the ? character can be used.

```
270,SVPROP=&1
\&1=:?
STATE VECTOR PROPAGATOR SELECTION FLAG
"CON"=CONIC,"AEG"=ANALYTIC EPHEMERIS GENERATOR
\&1:"AEG"
290,GPMPTET@&1
\&1:RESULT(1,3)
300,SUMTAB@&1
\&1:ICIRTAB
\:
```

The editor switches to the update mode when all required parameters have been provided.

Interface table editing is illustrated by construction of the IT for the OMS-2 circularization maneuver.

It would be easy to assume that the interface table is now finished. That is not the case, however. Values have been provided for all parameters which are "required" but this includes only those parameters which are <u>always needed</u> by the X Processor in question. In this particular case, there are also some optional parameters for which values will be needed <u>because of the particular way the user is applying the processor this time</u>. In this particular case, most of the optional parameters are needed because the "CIRC" maneuver option was selected. At this point, the user has two basic options. One is to wait and provide the values during actual execution -- each time the processor needs a value not provided in the table, the editor will be invoked automatically. The other is to provide the values now, before processor execution. To do this, the user might use the MODE command to get into the MISSING mode.

```
\:MODE,M
```

The system will then step through the optional parameters which are missing from the IT, eventually reaching those of interest to the user in this case.

```
\240,DENSMODL=:!!DENSMD
\250,SOLAFLUX=:!!SOLRMD
\260,GEOPOTEN=:?
 GEOPOTENTIAL MODEL
\260,GEOPOTEN=:&1
\:
```

With the exception that the user has intentionally deferred the definition of an input date element defining the geopotential model, the IT is now complete. It can be kept (in the AWA) simply by a normal exit from the editor

```
\:EXIT
%:
```

The table is now available for actual use by the GPMP processor.

25 Sequence Table Editing, NUMBER Command, NOTE Command

Because sequence tables are the only tables that can contain Executive commands, some unique ST editing capabilities are provided.

The way in which FDS prompts the user depends on the mode and the state of the ST entry. When the editor is invoked with the name of an existing sequence table, it starts in the INCOMPLETE mode, giving the user a chance to complete any incomplete entries. (Of course, if there are no incomplete entries, the editor automatically switches to the update mode.) Each incomplete entry generates a prompt which shows the entire incomplete entry and gives the user an opportunity to specify one missing value. The user is prompted for each succeeding nonexistent value of the entry, until all such values have been addressed:

```
120,PERFORM,GPMP,&ITNAME
#&ITNAME:OMS2
```

In this example, the user has provided the name (OMS2) of the interface table which is to be used in connection with one execution of the GPMP processor. A similar prompt is provided in the ALL mode whenever an incomplete entry is encountered. Whenever the user substitutes information for a placeholder, the substituted information is checked to see that it follows the syntax rules for the particular statement. In the UPDATE mode, the prompt consists simply of the ST editor symbol:

```
#:
```

MODE	USE
UPDATE	edit, list, change, etc. with Executive directives
CREATE	create new tables, add entries at end of existing table
INCOMPLETE	view and change incomplete entries
ALL	view each entry from start of table

25-1. Sequence Table editing can be done in any of four modes.

When the editor is invoked without naming an existing ST, it starts in the CREATE mode. This mode can also be entered via the mode command, of course. In the create mode, the editor allows the user to add new statements at the end of the table. The editor automatically generates the entry numbers, and the user need only type the statement itself:

```
#110,:PERFORM,GPMP,OMS1
#120,:PERFORM,GPMP,&ITNAME
```

In this example, the user has typed in commands which will later cause the execution of the GPMP processor for both OMS-1 and OMS-2 burns. The naming of the interface table for the OMS-2 burn has been deferred by the user.

In the UPDATE mode, the user can obtain simple tutorial information about the statements which may appear in a sequence table. By typing a question mark (?) character, the user can obtain a list of legal ST statements. These statements include all Executive directives except CLEAR and OFF, and also include DO LOOP, LOOP END, IF and ELSE. By typing a statement keyword followed by a question mark (e.g., STORE?), the user can obtain information about the syntax of a particular statement.

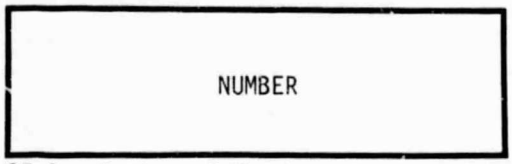
The sequence table editing function is used to create new sequence tables, and to modify existing STs. Sequence tables are the only table type that can contain commands to the Executive. The NUMBER directive can be used to renumber the table which is being edited.

NUMBER

The NOTE statement allows the user to store textual comments within STs.

NOTE,"text"

Two FDS commands are especially associated with sequence tables and their editing. The NUMBER directive allows the FDS user to renumber the entries in a sequence table or Y-data table which is being edited. The entries retain their original order, but are renumbered, beginning with 10 and incrementing by 10 for each subsequent entry. The NUMBER directive can be executed only when the editor is in the update mode.

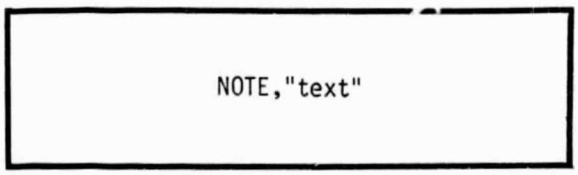


25-2. Syntax of the NUMBER directive.

The NOTE statement allows the FDS user to add comments to sequence tables. These comments are not executed, but are used for clarifying the purpose of the sequence table or to provide tutorial information. For example, the ST might be divided into segments, each identifed by a NOTE:

110,NOTE, "THE FOLLOWING STATEMENTS ESTABLISH A 200 NM. ORBIT"

NOTE statements are simply stored in the table like any other statements. They are always displayed when an ST is listed or executed in the SEMIAUTOMATIC mode. When a ST is executed in the AUTOMATIC or BATCH mode, the comments are displayed only if the appropriate trace option is specified. The text portion of the comment is bracketed by quotation marks ("), and may be up to 1600 characters long (counting control-C characters).



25-3. Syntax of the NOTE statement.

26 AN EXAMPLE OF SEQUENCE TABLE EDITING

The preparation of the basic sequence table for trajectory planning of the example problem provides a good, fairly typical example of ST editing.

A previous module, "SEQUENCE TABLES", presented a trajectory planning ST as an example. This module will discuss the use of the editor to construct that ST. Assuming that the user starts from scratch, rather than with an existing ST, the command which starts the editing process is

```
%:EDIT,,EXP1-ST
```

Since no old table name is given in this command, the editor starts in the CREATE mode. In this mode, the editor generates the entry numbers and prompts the user for the contents of each entry.

```
#10,:NOTE,"INITIALIZATION"
#20,:PERFORM,BASTM,T1
#30,:PERFORM,PHYDM,HOWBIG
#40,:ALLOCATE,INVEC(30,20)-R
#50,:ALLOCATE,RESULT(30,20)-R
#60,:ALLOCATE,HIALT(30,20)-R
#70,:NOTE,"ASCENT"
#80,:PERFORM,ASCENT,&ITNAME
```

Note that the user has deferred the decision concerning the IT name for the ASCENT processor.

By using the ? character, the user can obtain helpful information. For example, the response

```
#90,:?
```

will cause FDS to list all the statements which can appear in a sequence table, while the response

```
#90,:ASSIGN?
```

allows the user to obtain information about the syntax of a particular statement (in this case, ASSIGN). When the initial input of the sequence table has been completed, the user can input the ST editor symbol to change to the UPDATE mode.

```
#320,:PERFORM,GTRACK,GTK
#330,:PERFORM,FPD,FLTP
#340,:#
#:
```

Now the table can be LISTed and edited further using CHANGE, INSERT, and other commands discussed later. If the user changes to the INCOMPLETE mode at this time, FDS will provide prompts for any information which the user deferred.

```
#:MODE,I
80,PERFORM,ASCENT,&ITNAME
#&ITNAME:UP
#:
```

Alternatively, the user might store the table in this incomplete form, in which case the editor would automatically start in the INCOMPLETE mode in any subsequent editing session.

Sequence table editing is illustrated by construction of a basic sequence table for the trajectory design portion of the example problem.

When the editing session is finished, the user can EXIT normally,

```
#:EXIT
%:
```

In which case the generated table will be stored in the AWA under the name EXP1. Alternatively, the editing results can be thrown away by entering the Executive prompt symbol (%).

```
#:%
%:
```

In this case, the edited table (EXP1-ST) is deleted. Care must be used in aborting the editor in this way, in order to avoid inadvertent deletion of needed information.

27 ABSTRACT Command (Creating, Replacing, and Displaying Abstracts)

The ABSTRACT command displays the date/time tag and an abstract for any specified table, DE, DRDE, data base, YOD, document/segment, Xjob, Yjob, or processor, and allows the user to create or replace abstracts for personally owned files, data bases, etc.

The ABSTRACT command can be used to learn what information is present in tables, files, or data bases without viewing all of the information. An abstract is a textual description of the contents of an OBJECT. Included with the abstract is information on the date and time the object was last changed and the creator of the table, file, data base, or processor. The user can add a new abstract where none exists or replace an existing abstract for personally owned objects or for another user's PDB, if the user has access to it. MDBs, X and Y Processors, and document/segments have abstracts that are maintained by FDS support personnel. These abstracts can be displayed, but not modified.

```
ABSTRACT-device code,object,"text of abstract"
```

27-1. Syntax of the ABSTRACT command.

OBJECT is an argument used in some commands to reference a file, data base, table, or processor. The OBJECT consists of an OBJECT NAME, a DATA CLASSIFICATION CODE, and, in some cases, an ACCESS CODE. The object name is the name of the file or data base to be processed while the data classification code indicates the file or data base type. If the object is a PDB belonging to another user, that user's access code must also be provided. The access code is a two-character code; an access code is assigned to each FDS user.

```
Abstracts can be added, replaced, or displayed for these data items

     table name-class            TAB1-IT
     data element name-DE        STAT-DE
     DRDE name-DF                DSK-DF
     PDB name-PD                 PROB6-PD
     YOD file name-YJ            YDAT1-YJ
     X Job name-XJ               XDAT1-XJ

Abstracts can only be displayed for these data items

     MDB name-MD                 !!GLOCON-MD
     X Processor name-XP         XPYES-XP
     Y Processor name-YP         MNSTR-YP
     document/segment name-DS    CFPTRJ-DS

To display the abstract for another user's PDB, use that user's access code

     PDB name-PD-access code     FILE1-PD-UC
```

27-2. The OBJECT field is used to indicate a file, data base, or processor.

An abstract can be associated with all tables, files, data bases, and processors. The abstract is a brief, textual description of the contents of a data item. By adding abstracts to each data item that is created, users will be able to identify the contents of that item more readily. The ABSTRACT command is used to create, replace, and display abstracts.

ABSTRACT-device code,object,"text of abstract"

- ST	sequence table
- IT	interface table
- LT	linkage table
- YT	Y-data table
- DE	data element
- DF	DRDE
- PD	personal data base
- MD	master data base
- YJ	YOD or Y-job
- XJ	X-job
- XP	X processor
- YP	Y processor
- DS	document/segment

27-3. DATA CLASSIFICATION CODES indicate the type of data to be processed.

The DEVICE ID is used to designate the output device for display purposes. Output from the ABSTRACT, and other commands, can be displayed either at the user's terminal or on the system line printer. If no device identifier is specified, the line printer is assumed for the BATCH mode and the terminal is assumed for interactive sessions.

The "TEXT OF THE ABSTRACT" is a character string of up to 256 characters. It must be enclosed in quotation marks ("). If the abstract to be entered requires more than one line of input, entering a carriage return will cause the prompt CONTINUE: to appear, and the user can continue entering the text. The text is terminated with a quotation mark ("). Typing Control-C within the text causes a new line in the displayed abstract.

To display the abstract for an interface table (without update):

```
ABSTRACT, TAB1-IT
```

To display the abstract for another user's PDB:

```
ABSTRACT, PDB1-PD-KP
```

To replace and then display the abstract for a data element:

```
ABSTRACT, EVENTS-DE, "TRAJECTORY EVENTS THROUGH OMS-2"
```

27-4. Examples of use of the ABSTRACT command.

One of the uses of the ABSTRACT command is to obtain information about processors and MDBs provided by the system. In the example problem, one of the X Processors that will be used is GPMP. In order to display the abstract for this processor, the command is:

```
ABSTRACT,GPMP-XP
```

and the resulting display provides basic information about the nature and use of this X Processor.

The second major purpose of ABSTRACT is to allow the user to add information to a file which describes its nature and use. For the example problem, the user might create a sequence table called EXPL1 and add an abstract that identifies it as being related to the example problem. In this case, this command may be used:

```
$:ABSTRACT,EXPL1-ST,"THIS TABLE CONTAINS A SEQUENCE OF COMMANDS (CR)
CONTINUE:THAT SOLVES THE ® CFP PORTION OF THE EXAMPLE PROBLEM (CR)
CONTINUE:IN THE FDS USER'S GUIDE."
```

28 Command (Display Contents of Tables and Other Files)

The LIST command is used to display the contents of tables and files.

```
LIST-device ID,object name(span)-file type, ...
```

28-1. Syntax of the LIST command.

The LIST command allows the FDS user to display the contents of any table, DE, DRDE, or YOD file. Ordinarily, the user indicates one or more files to be listed, by name. If several files are named, they will be listed in the order given in the command. Files can be listed in their entirety, or the user can specify which line or lines of the file are to be listed. The user can also specify the device on which the listing is to be done. If no device ID is given, the listing is transmitted to the user's terminal. If the characters "-P" are added after the word LIST, the output will go to the system line printer.

```
Sequence table - ST
Interface table - IT
Linkage table - LT
Y-data table - YT
Data elements - DE
Disk-resident data elements - DF
YOD file - YJ
```

28-2. The OBJECT field is used to indicate a table or file.

The SPAN field is used to specify limits on the contents of tables or DEs to be listed. For tables and YOD files, the span specifies the entry numbers of the first and last table entries to be listed. For DEs, the span field consists of the subscripts of the items to be listed. If either the start or end of the span is not indicated, the first or last entries, respectively, are assumed. The user may specify individual subscripts or a range of subscripts or any combination of these two. Only entire DRDEs may be listed.

	MEANING	EXAMPLE
For tables and YODs	single entry number	100
	start entry number - end entry number	100-600
	start entry number -	100-
	- end entry number	-600
For DEs	single subscript	2
	start subscript - end subscript	2-6
	start subscript -	2-
	- end subscript	-6

28-3. The SPAN field sets boundaries on the information to be listed.

The LIST command is used to display the conterts of tables and files. Entire tables and files or only parts of tables and files can be listed. Ordinarily, the listings will use the same format as used in the table or file being listed. However, if a DE or DRDE uses the "free" data type, the user must indicate the format to be used.

LIST-device ID,object name(span)-file type,...

The LIST function may be used as a directive within a table editor. To list the table being edited, the user need only specify the span of the table to be listed:

```
LIST,(200-500)
```

If the span is omitted, the entire table is listed.

```
LIST
```

Any table or file can be listed, however, by using its name with the LIST function.

If a DE or DRDE consists of information that is stored as free data, the user must supply the format to be used. In this case, FDS will prompt the user to supply the format to be used to list the DE or DRDE. If no format is supplied, or if the LIST function is executed from the batch mode, a hexadecimal format is assumed.

Dialogue	Comments
LIST,BIG-DF	BIG has free data
FORMAT FOR NAME:	The user is prompted to supply the format, which may be: I: Integer R: Real D: Double precision Ch: Character (n=4,8,16,32,or 64) T: Time

28-4. The user supplies the format for free data.

In solving the example problem, the user constructed an interface table for the OMS-2 circularization burn (see "An Example of Interface Table Editing"). That entire IT, or sections of it, can be displayed using the LIST command. For example, by typing

```
LIST,OMS2(240-270)
```

The user can obtain the following display:

```
240  DENSMODL=!!DENSMD
250  SOLAFLUX=!!SOLRMD
270  SVPROP="AEG"
```

This display illustrates a situation in which the user must be aware of the circumstances. Entry number 260, a missing, optional parameter, is not displayed here. IT entries which are (1) optional, and (2) not filled in, are not printed when the LIST command is used. If they were, many missing entries would be printed even through they are not relevant to the case in question. Thus, the LIST command can be used to show all entries which have values (including all required entries even if incomplete), but only the EDIT command allows the user to see those additional entries which represent missing optional variables.

29 GET, STORE, APPEND, SUBSTITUTE Commands (Permanent Storage and Retrieval of Files)

The GET command, STORE command, APPEND command, and SUBSTITUTE command are used to obtain files from permanent data bases, and to store files in such data bases.

The GET command allows a user to retrieve all or selected portions of a PDB or MDB. The files thus retrieved are added to the current contents of the AWA. In using the command, the user gives the name of the data base which contains the files now. This may be an MDB (all users have access to these) or any PDB for which the user knows the name and access code. The user must also specify which file or files are to be moved.

GET,data base name-type-access code,files to be moved
STORE,PDB name-PD-access code,files to be moved
APPEND,PDB name-access code, files to be moved
SUBSTITUTE,PDB name-access code,files to be moved

29-1. Syntax of the GET, STORE, APPEND and SUBSTITUTE commands.

The user may specify that all files in the data base are to be moved to the AWA, or may move whole classes of files, or individual files. If the "files to be moved" portion of the command is omitted, as in the command

```
GET,EXPL-PD
```

then all files are moved. Otherwise, a list of individual file names, file type codes, or both, can be used. Ordinarily, a single file is specified by giving both its name and type, as in the command

```
GET,EXPL-PD,OMS2-IT
```

If only a name is given, the file is assumed to be a DE. If only a type code is given (preceded by a hyphen), as in the command

```
GET,EXPL-PD,-IT
```

all files of the indicated type are moved. These same conventions apply to the STORE and APPEND commands, which move files from the AWA to permanent data bases.

	If "files to be moved" is:	The effect is:
1	omitted	all files in the source data base (or AWA) are moved.
2	file type with no file name (e.g., -ST)	all files of the specified type are moved.
3	file name with no type (e.g., ABXYZ)	a data element with the specified name is moved.
4	file name with type (e.g., OMS2-IT)	the specified file is moved.
5	a list of specifications of forms 2, 3, and 4, above (e.g., ABXYZ, -ST, OMS2-IT)	each file or group so specified is moved.

29-2. The "files to be moved" are specified in the same way for all four commands.

Transferring data from a PDB or MDB into the AWA is done with the GET command. The files retrieved are added to the current contents of the AWA.

GET,data base name-type-access code,files to be moved

Storage of AWA files into a PDB is done using the STORE or APPEND commands. STORE creates a new PDB, and places the indicated AWA files in it.

STORE,PDB name-PD-access code,files to be moved

APPEND adds the indicated files to an existing PDB.

APPEND,PDB name-access code,files to be moved

SUBSTITUTE purges all files in the named PDB, then stores the indicated files there.

SUBSTITUTE,PDB name-access code,files to be moved

The APPEND command is similar to GET, but moves the files in the opposite direction, from AWA to PDB. The user specifies the name of the existing personal data base to which files are to be moved from the AWA. The indicated files are "appended" to the previous contents of that PDB. If any files to be moved have the same name as files already in the PDB, an error message is generated and those particular files are not moved. Since the FDS user is not allowed to modify the contents of master data bases, only a PDB can be named in this command.

The STORE command is similar to APPEND, but creates a new PDB and moves the files from the AWA into it. This command is the mechanism for creating new PDBs, and is the ordinary way of saving all the information associated with a particular flight design. Although several approaches can be used, it may be easiest to delete all extraneous files from the AWA, and simply store the rest in a new PDB by using the STORE command with no "files to be saved" specification. It should be noted that the PDB must have a different name than any other PDB having the same access code. If this condition is not satisfied, an error message is generated and no files are moved. A STORE command will also generate an error if the user already has the maximum number of PDBs. In this case, the user must delete an existing PDB, APPEND the files to an existing PDB, or create a PDB under another access code.

```
Get all interface tables from a PDB:
     GET,FLIGHT1-PD,-IT

Get entire MDB:
     GET,CONSTANT-MD

Save entire AWA in new PDB:
     STORE,FLIGHT2-PD

Add specified files to existing PDB:
     APPEND,FLIGHT1,OMS2-IT,TRAJECT-ST

Replace existing PDB with current
contents of AWA:
     SUBSTITUTE,FLIGHT2
```

29-3. Examples of GET, STORE, APPEND, and SUBSTITUTE commands.

The SUBSTITUTE command is similar to STORE, but places AWA files into an existing PDB after clearing it. All files previously in the PDB are purged, but the name and abstract of the PDB are retained. The indicated AWA files (or the entire AWA) is then stored in the PDB.

In solving new flight design problems, it may be helpful to develop a "library" of standard flight designs. Flight designs in such a library can then be used as a starting point when new, similar flights are to be planned. If we suppose that a PDB ("GEOSYNCH") already exists for launching a geosynchronous payload, that flight design might easily provide a good starting point for the example problem. To load the PDB into the AWA, the user might type:

GET,GEOSYNCH-PD-AA

The user would then modify the GEOSYNCH flight design as necessary, adding the additional payload, etc. When completed, the resulting flight design information might be saved as PDB EXPL:

STORE,EXPL-PD.

30 TOC Command (List Names of Files, Data Bases, etc.)

The TOC (Table of Contents) command allows the user to display a list of files in the the AWA or a data base or to list X or Y jobs, X or Y processors, or available data bases.

The basic purpose of the TOC command is to list the contents of the AWA or of any data base to which the user has access. Thus, the command

```
TOC
```

produces a table of contents for the AWA, while the command

```
TOC,,EXPL-PD
```

produces a table of contents for the user's PDB, called EXPL. Tables of contents for MDBs can also be produced. TOCs will be displayed on the user's terminal, unless the system line printer is specified, as

```
TOC-P,,EXPL-PD
```

When the TOC command is executed in the batch mode the display automatically goes to the printer.

TOC-device ID,class, data base name-classification code-access code

30-1. Syntax of the TOC command.

By specifying the file type, the user can obtain a list of the files of a particular class which are contained in the AWA or in a specified data base. Standard file type codes are used for this, so that

```
TOC,ST
```

produces a list of sequence tables in the AWA, while

```
TOC,IT,EXPL-PD
```

produces a list of interface tables in the PDB named EXPL.

Other class codes can be used to list the user's jobs, or available data bases, processors, or document segments. For example, the command

```
TOC,XP
```

displays the names of all available X Processors.

Class Codes	Meaning
-ST	sequence tables
-IT	interface tables
-LT	linkage tables
-YT	Y-data tables
-DE	data elements
-DF	DRDE files
-PD	personal data base files
-MD	master data base files
-YJ	YOD files or Y-jobs
-XJ	X-jobs
-XP	X-processors
-YP	Y-processors
-DS	document/segments

30-2. The TOC command lists a variety of objects depending on the "class" selected.

A table of contents (TOC) is associated with the AWA and each PDB and MDB. The table of contents indicates the name, type, and size of each AWA or data base entry. The TOC command is used to display a table of contents and also to provide a list of X or Y jobs or processors, or a list of available PDBs or MDBs.

TOC-device ID,class,data base name-classification code-access code

In solving the example problem, the user will accumulate a number of interface tables, data elements, etc., which will reside in the AWA until stored in a PDB. Some of these elements (e.g., interface tables) are constructed by the user, while others (e.g., data elements) are constructed as the Flight Design System performs the commands in the sequence table. When the sequence table has been executed, the user can display the entire contents of the AWA via the command

TOC

and obtain a display similar to the following

```
!TOC
                                TABLE OF CONTENTS
FOR THE AWA                                        DATE = 06/02/80  TIME = 15/23/15
-------------------------------------------------------------------------------

  INTERFACE TABLES
   NAME     ABS   DATE       TIME       AC   SIZE
--------    ---  --------   --------    --  ------
BASE1           04/12/80   11/47/36    WE     141
PCOAST1                                        384
UNITS           04/02/80   17/25/20    WE     390

  DATA ELEMENTS
   NAME     ABS   DATE       TIME       AC   SIZE   TYPE   ROW   COL
--------    ---  --------   --------    --  ------  ----   ----  ----
!BDATE          04/12/80   11/47/36    WE     164  M039     50
!SCOEF          04/12/80   11/47/36    WE      32  M042     15
!SESCON         04/02/80   17/25/20    WE     288  M041     72
SV01            04/02/80   17/25/20    WE      56  F        28
SVIN            05/16/80   18/05/05    WE      56  M072     15
TETAB           05/16/80   18/05/05    WE     240  M073     42      2

  DRDE FILES
   NAME     ABS   DATE       TIME       AC   SIZE   TYPE   LRL   FT
--------    ---  --------   --------    --  ------  ----   ----  --
ISFILE          05/16/80   18/05/05    WE       1  M072     56   I

AWA SIZE = 16000     FREE = 14408  (HALFWORDS)
DWA SIZE =   400     FREE =   377  (SECTORS)
COMPACTIONS =       0,       0,       0,       0,       0,       0
```

31 CLEAR, DELETE Commands (Eliminating Files from Storage)

The CLEAR and DELETE commands are used to eliminate unneeded information from storage.

The CLEAR command purges all files from the user's Active Work Area. This results in the deletion of all interface tables, sequence tables, linkage tables, Y-Data tables, data elements, and disk-resident data elements. Because MDBs, PDBs, and YODs do not reside in the AWA, they are not affected. The CLEAR command can be used only at the Executive level (i.e., with a prompt of %:).

31-1. Syntax of the CLEAR command.

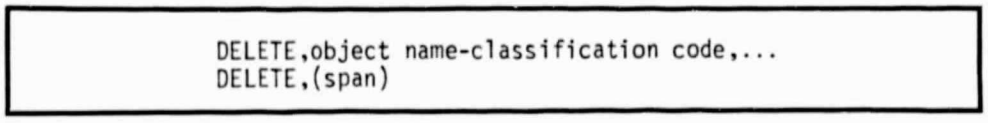
The DELETE command can be used to eliminate individual files or whole classes of files by specifying file names and/or types. For example, the command

```
DELETE,OMS2-iT
```

causes the interface table "OMS2" to be discarded from the AWA. The space previously occupied by the deleted file is thus made available for other AWA files. The command

```
DELETE,-DE,-DF
```

results in the elimination of all data elements and DRDEs (file type DF) from the AWA. This command might commonly be given before execution of a sequence table which constructs these elements, and might precede storage of the AWA as a PDB via the STORE command.



31-2. The DELETE command has two forms.

Files which are in use cannot be deleted. For example, a sequence table cannot be deleted while it is executing, as might happen if it contained a statement such as

```
DELETE,-ST
```

Similarly, a YOD cannot be deleted until the corresponding Y job has finished executing.

The DELETE command can be used to eliminate PDBs by name. PDBs cannot be deleted as a class, but the user can delete his own PDBs by specifying their name and class codes, as

```
DELETE,EXPL-PD,OTHER-PD
```

PDBs cannot be deleted if they belong to access codes other than that used to log on.

By specifying first and last entry numbers ("span"), the user can delete a portion of a table currently being edited. Thus, the command

```
DELETE,(20-40)
```

deletes entries 20 through 40 of the table which the user is editing. If a DELETE command specifies a portion of an interface table, that portion is merely cleared of values, but the variable names remain, since entries cannot be deleted from an IT.

The CLEAR command is used to eliminate all files from the AWA.

```
CLEAR
```

The DELETE command is used to eliminate individual files or whole file classes from the AWA, to eliminate individual PDBs, or to delete specified portions of AWA files.

```
DELETE,object name(span)-classification code,...
DELETE,(span)
```

OBJECT	MEANING
1. Name of PDB (e.g., EXPL-PD)	User's PDB with specified name is deleted.
2. Name of file with file type (e.g., OMS1-IT)	Named file is deleted from AWA.
3. Name of file without file type (e.g., RESULT)	Named data element is deleted from AWA.
4. Span only (e.g., (50-60))	Indicated span of entries is deleted from file currently being edited.
5. File type only (e.g., -YD)	All files of indicated type are deleted from AWA.
6. List of objects of types 1-5. (e.g., -YD,MINE-PD, (20-60))	Each object is deleted in turn.

31-3. The DELETE command can be applied to a variety of objects.

A typical work session begins with the loading of a PDB, as by the command

```
GET,EXPL-PD
```

in order to restore the conditions which existed at the end of the user's previous session. In this case, the GET command restored the PDB into which we placed all the files associated with the example problem. During the course of the work session, the user may modify some of these files and wish to save the updated versions. One approach might be

```
DELETE,EXPL-PD
STORE,EXPL
```

which saves the entire current contents of the AWA. To conserve disk space, the user might prefer to eliminate those files which can easily be reconstructed. In this case, the command sequence might be

```
DELETE,EXPL-PD,-DE,-DF
STORE,EXPL
```

When this operation is completed, the user can sign off or begin a new problem, as with the commands

```
CLEAR
GET,NEWPROB-PD
```

which clears the AWA of the remnants of the old problem before obtaining information associated with the new.

32 COPY, RENAME Commands (File Duplication and Name Changing)

The COPY command allows the FDS user to duplicate and rename a table, DE, DRDE, or PDB, while RENAME only changes the name of the original.

The COPY command must indicate the object to be copied, and the name to be assigned to the new copy. The object may be a sequence table, interface table, linkage table, or Y-data table, as in

```
COPY,TRAJECT-ST,SAVET
```

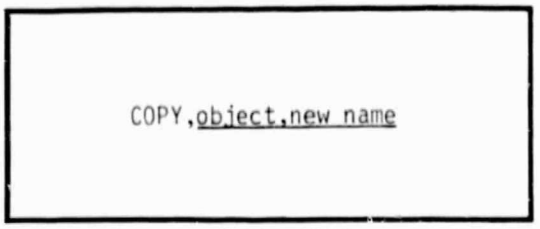
or a data element or DRDE, as in

```
COPY,RESULT,RESULT2
COPY,BIGFILE-DF,WORKCOPY
```

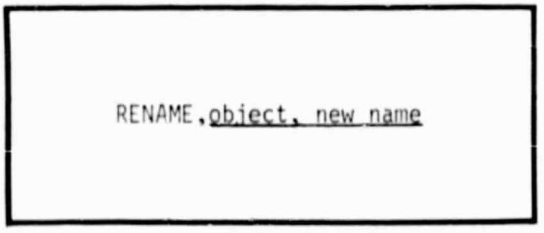
or a PDB, as in

```
COPY,HISPDB-PD-HE,MYPDB.
```

In the case of PDBs, the new copy will have the access code under which the user logged on, regardless of the access code of the original PDB. All other objects are copied from the AWA to the AWA. In every case, the new name must not conflict with the name of any existing object of the same class.



32-1. Syntax of the COPY command.



32-2. Syntax of the RENAME command.

The RENAME command merely changes the name of the original object, but does not make a copy. The objects which can be renamed are the same as those which can be copied, except that the user may rename only his own PDBs. Thus, the user can RENAME tables and DRDEs,

```
RENAME,BUILD-IT,OMS2
```

data elements

```
RENAME,DATA,SAVEDATA
```

or PDBs

```
RENAME,EXPL-PD,TEMPSAVE.
```

The new name must not conflict with any existing object of the same class.

With both COPY and RENAME commands, the "new" object is given a "current" date/time/user ID tag. This information, which is associated with every file or PDB, can be useful in determining when files were constructed or last modified. Actually, the date/time information is set to zero to indicate "current" information and is only replaced by a real date and time when the file is stored in a PDB. Thus, a date/time tag of zero (in a TOC dispay, for example), indicates a file which has been generated or modified during the current user session. If the user makes use of the date/time information, it should be kept in mind that even renaming the file causes the date and time to be updated.

It is important to note that RENAME changes only the name under which the file is stored. Any references to the old file name which are made in sequence tables, interface tables, etc., are left unaltered. It is the user's responsibility to insure that such references point to the correct file.

The COPY command is used to copy and rename tables, DEs, DRDEs, and PDBs.

COPY,object,new name

The RENAME command changes the name of a table, DE, DRDE, or PDB.

RENAME,object,new name

Although COPY and RENAME operations can be done for a variety of purposes, a particularly common reason is the saving of files or data bases which are about to be changed. In the example problem, the user works with a PDB called EXPL, which is assumed to be loaded (via the GET command) at the end of each session. Thus, a session might include the commands

```
GET,EXPL-PD
...
SUBSTITUTE,EXPL-PD
```

The SUBSTITUTE command is used, instead of STORE, because the STORE command cannot construct a PDB whose name conflicts with one already in existence. It is easy to imagine circumstances in which the user wants to save this PDB, so as to have a back-up copy of the PDB as it existed before the session. In this case, the commands,

```
RENAME,EXPL-PD,BACKUP
STORE,EXPL-PD
```

might be used in place of SUBSTITUTE.

A similar situation sometimes exists with respect to data elements or DRDEs. In the example problem, the TRAJECT sequence table builds, among other things, a DE called "RESULT". When modifications are made to TRAJECT, or to Interface Tables, etc., which are used in executing TRAJECT, it might be desirable to save a copy of RESULT. The new RESULT can then be compared with the saved copy in order to observe the effects of the changes. Since TRAJECT builds, in effect, a new copy of RESULT each time it is executed, the RENAME command can be used:

```
RENAME,RESULT,SAVECOPY
```

The reader will recall that the file type can be omitted when the file is a DE.

The COPY command is the right mechanism to use when the file is built directly by the user. Suppose, for instance, that the user is about to modify OMS2-IT, and wants to keep an unmodified version. An appropriate command might be

```
COPY,OMS2-IT,SAVEOMS2
```

The user can then edit the OMS2 file as desired, as by the command

```
EDIT,OMS2-IT
```

or the CHANGE command (discussed later). It should be noted that the EDIT command accomplishes a similar, but not identical, function. The command

```
EDIT,OMS2-IT,SAVEOMS2
```

also results in the existence of two copies, but the file called "OMS2" is the unmodified version. Another approach to the COPY and EDIT sequence above is

```
RENAME,OMS2-IT,SAVECOPY
EDIT,SAVECOPY-IT,OMS2
```

Another common use of RENAME is the situation in which the user wants to GET files from two or more data bases, but some files have the same name. Suppose, for instance, that the user is beginning the solution of the example problem and wants to construct the TRAJECT file by merging two existing TRAJECT files, one of which resides in the GEOSYNCH PDB and one in the CIRC200 PDB. The commands used might include

```
CLEAR
GET,GEOSYNCH-PD-XY
RENAME,TRAJECT-ST,TEMP
GET,CIRC200,TRAJECT-ST
```

followed by the merging of the two files (see INSERT command).

33 CHANGE, INSERT Commands (Modifying and Merging)

The CHANGE command is used to modify abstracts or table entries, while INSERT merges the contents of two tables.

The CHANGE command can be used to alter information in a table which is being edited. The command must specify which entry or entries are to be changed. Within the specified span of entries, a search will be made for all occurrences of the indicated "current data". Each occurrence will be replaced by the "new data". Current and new data may be object names, as in the command

```
CHANGE,(300-300),ITAB1,ITABA
```

which replaces an interface table name in line 300 of the table currently being edited. In STs only, they may be numbers, as in

```
CHANGE,(10-),14,286.3E-14
```

(which searches all entries from 10 to the end of the file). Or (in STs) they may be special symbols

```
CHANGE,(1285-1286),+,-
```

except that the comma and quotation mark (") may not be used in this way. The comma is used in the command itself, and quotation marks are reserved as delimiters for string literals, which are also allowed, as in

```
CHANGE,(-65),"GPMP,OM","FLT,FP".
```

In STs, the "current data" and "new data" need not be of the same type. Commands like

```
CHANGE,(287-290),!!INPUT,26
```

are acceptable. Furthermore, several whole elements of these types may be strung together in order to unambiguously identify the desired current data or to replace this information with several elements, as in the command

```
CHANGE,(20-20),OMS2-ST,OMS2-IT
```

It is the user's responsibility to insure that the new data satisfy the requirements of the table type being modified. The syntax of the table is not checked until the table is actually used.

To alter abstracts:

CHANGE,object name-classification code,current data,new data

To alter table being edited:

CHANGE,(span),current-data,new data

33-1. The CHANGE command has two forms.

ST = sequence table
IT = interface table
LT = linkage table
YT = Y-data table
DE = data element
DF = disk-resident data element
PD = personal data base
XJ = X job
XJ = YOD file or Y job

33-2. CHANGE can alter the abstract of any object type which has abstracts.

The CHANGE command can also be used to alter the contents of any abstract. In this case, the command must name the object and its type, but does not, of course, include entry numbers. For example, the command

```
CHANGE,EXPL-PD,"LUNCH","LAUNCH"
```

searches the entire abstract of the EXPL PDB for occurrences of the string "LUNCH" (without quotation marks) and relaces each such string with "LAUNCH".

The CHANGE command is used to modify abstracts or to modify a table which is currently being edited.

CHANGE,object name-classification code,current data,new data
or CHANGE,(span),current data,new data

The INSERT command is used to combine all or a selected portion of a specified table with another table, of the same class, which is currently being edited.

INSERT,source table name(span),insert location

When used with sequence tables or Y-data tables, the INSERT command allows all or part of a table to be inserted into the table being edited. The inserted information may be taken from some other table, as in the command

INSERT,TRAJECT-ST

which inserts the entire TRAJECT sequence table at the end of the sequence table which is currently beng edited. Alternatively, information which is already in the current table can be inserted in the same table by omitting the name of the source table. This is useful when a particular segment of the table is to be repeated, perhaps with small changes, somewhere else in the table. For example, entries 20-40 of the current table are duplicated at the end of the table by the command

INSERT,(20-40)

The command can also specify that the inserted information is to be placed after a particular entry number. Thus the command

INSERT,SAVETRAJ(100-200)-ST,70

inserts line 100-200 of the SAVETRAJ sequence table after entry 70 of the sequence table which is being edited. To insert at the beginning of the table, the user indicates that the insertion is to be made after the (nonexistent) entry zero, as in

INSERT,PRELIM-ST,0.

Whenever INSERT is used with a sequence table or Y-data table, the table entries are renumbered (10, 20, 30, etc.). Incidentally, FDS does not allow the INSERT operation to occur when the source table and edited table are not of the same type (this operation wouldn't make sense, anyway).

INSERT,source table name(span)-file type,insert location

33-3. Syntax of the INSERT command.

With interface tables and linkage tables, INSERT causes a logical merging of the two tables. Because the structure of these tables is fixed, it is not possible to insert information between two existing entries. Thus, no insert location is allowed. For these tables, the table being edited is retained intact, except that all existing parameter values in the source table replace those in the edited table. That is, any entry in the source table which has a value will replace the corresonding entry in the edited table, whether it already had a value or not.

If the table being edited is	and the source table is	then the command	yields the result
Sequence table: 10,PERFORM,PA,TA 20,PERFORM,PB,TB	B-ST: 10,STORE 20,LIST,DE1-DE	INSERT,B,10	10,PERFORM,PA,TA 20,STORE 30,LIST,DE1-DE 40,PERFORM,AB,TB
Interface table: 40,D=BETA 60,F=1,2	A-IT: 10,A=1,2,3 60,F=ALPHA 80,H=1,&,2	INSERT,A	10,A=1,2,3 40,D=BETA 60,F=ALPHA 80,H=1,&,2

33-4. INSERT has different effects for different table types.

34 CROSS REFERENCE, FIND Commands (Locating Information in Tables)

The CROSS REFERENCE command finds all references to specified objects within AWA-resident tables, while FIND searches one particular table or table segment for items which need not be object names.

The CROSS REFERENCE command is used to locate all references to a particular table or other object. Suppose, for example, that the user wishes to substitute interface table OMS2A for OMS2 in certain instances, but not in others. It might be helpful to use the CROSS REFERENCE command to locate all references to OMS2 in existing sequence tables in the AWA:

```
CROSS REFERENCE,-ST,OMS2-IT
```

This causes a cross reference display to be presented on the user's terminal, showing each instance, in any sequence table, of a reference to OMS2. The command can be used to search only a particular table, as in

```
CROSS REFERENCE,TRAJECT-ST,OMS2-IT
```

which searches only the TRAJECT sequence table. When searching for references to data elements, DRDEs, or X Processors, the user may prefer to allow all AWA tables to be searched, regardless of type. DEs, for example, can be referred to from any table type. This is done by simply omitting both table name and type from the command:

```
CROSS REFERENCE,,RESULT-DE
```

By adding the line printer device code (-P) to the command, the user can cause the display to go to the system line printer:

```
CROSS REFERENCE-P,,GPMP-XP
```

CROSS REFERENCE-device ID,searched table name-file type,reference name-class

34-1. Syntax of the CROSS REFERENCE command.

CROSS REFERENCE can be used to search for a variety of object types. The user can search for any table (interface table, sequence table, Y-data table, or linkage table), as well as data elements, DRDEs, data bases (PDBs and MDBs), X Processors, YOD files, and document/segments.

If the item searched for is:	then the following table types may be searched.			
	ST	IT	YD	LT
Sequence table (ST)	X			
Interface table (IT)	X			
Y-data table (YD)	X			
Data element (DE)	X	X	X	X
Disk-resident data element (DF)	X	X		X
Personal data base (PD)	X			
Master data base (MD)	X			
YOD file (YJ)	X			
X Processor (XP)	X	X		
Document/segment				X

34-2. Not all search and reference combinations are valid in CROSS REFERENCE commands.

The CROSS REFERENCE command is used to search AWA-resident tables for references to tables, DEs, DRDEs, data bases, YOD files, X Processors, and document/segments.

CROSS REFERENCE-device ID,searched table name-file type,reference name-class

The FIND command searches an individual table for specific names, numbers, strings, or special symbols.

FIND,searched table name(span)-file type,data

The FIND command is somewhat similar, but can search for other data. Each entry, thus found, is displayed on the user's terminal. The search data can consist of object names, as in CROSS REFERENCE, but can also include numbers

```
FIND,ITAB-IT,-27.6
```

or, in sequence tables, qualifiers (classification codes, device codes, and access codes, which are normally preceded in use by a hyphen). FIND can search for individual symbols

```
FIND,ISEQ-ST,+
```

as well as character string segments

```
FIND,ISEQ-ST,"SPIDPO SUPPORT.  ON"
```

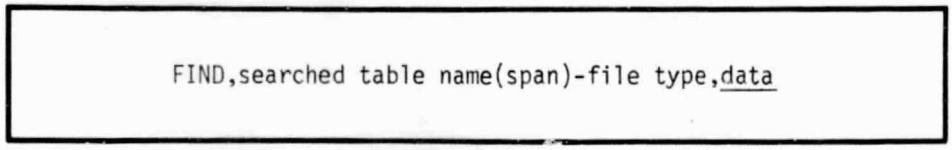
Furthermore, several components can be strung together in order to uniquely identify the information sought, as in the command

```
FIND,ISEQ-ST,ABC+10.
```

These must be whole components, however. The above command would not find an entry containing the group

```
BABC+10
```

since it looks for the whole name "ABC" followed by a plus sign and the number 10. The search data may not include a comma (unless it is part of a string literal), since the comma would be taken as part of the syntax of the command, rather than as part of the search data. And, of course, quotation marks may be used only as part of a string literal. A command such as FIND,ISEQ-ST,END." is not allowed. Percent (%) and semi-colon (;) characters are also not allowed, except within a string literal.



34-3. Syntax of the FIND command.

The FIND command can be used to search individual tables or table segments. Multiple tables cannot be searched with a single FIND command, as they were with CROSS REFERENCE. If the table to be searched is currently being edited, it need not be named. Thus,

```
FIND,,OMS2-IT
```

searches the table now being edited for references to OMS2-IT. The span can be limited by specifying starting or ending numbers in the usual way, as in the commands

```
FIND,(-500),OMS2-IT
FIND,TRAJECT(300-500)-ST,OMS2
```

35 ATTRIBUTES, PERFORM Commands (X Processor Properties and Execution)

The ATTRIBUTES command displays the attributes of all or selected parameters of an X Processor (or document/segment), while PERFORM causes the X Processor to be executed.

The ATTRIBUTES command allows the FDS user to display information about the input/output parameters of an X Processor. The user may specify particular parameters, or may obtain an attributes display for all the parameters associated with the processor. Thus, the command

```
ATTRIBUTES,GPMP-XP,APSOPT,SUMTABLE
```

produces the following display:

```
ATTRIBUTES DISPLAY FOR PROCESSOR GPMP          VERSION 6
ENTRY   KEYWORD   SIZE   ROW   COL   CLASS   TYPE   I/O   R/O
   19   APSOPT       2     1         DE      C4      I     0
   39   SUMTABLE   368     4    46    DE      F       0     R
```

This display indicates that entry no. 19 is for parameter APSOPT. APSOPT occupies 2 physical half-words and is a single logical entity (an array of size 1, if you like). It is defined as a data element (CLASS = DE) containing character data of up to 4 characters (TYPE = C4). It is an input parameter (I/O = I) and is optional (R/O = O). Entry 39 is for parameter SUMTABLE. This is a two-dimensional array (4x46 elements) occupying 368 half-words of storage. It is a data element and contains free data. It is an output parameter and is required. This particular display was obtained because the command specified the parameters APSOPT and SUMTABLE. If no parameters are specified, as in the command

```
ATTRIBUTES,GPMP-XP
```

then all parameter attributes are displayed. Like most display-generating commands, this one can be made to produce its output on the system line printer, as in the command

```
ATTRIBUTES-P,GPMP-XP
```

If an interface table is currently being edited, the attributes of the corresponding X Processor can easily be displayed. For example, the user may be editing the OMS2 table for processor GPMP, and be prompted as follows:

```
\24,SVPROP=:
```

Responding with a question mark,

```
\24,SVPROP=:?
```

results in the display of information about the nature of the parameter, but does not provide attributes information. Thus, the user may not know the size of an array, or, in this case, whether or not the parameter is required. By entering the update mode and typing an ATTRIBUTES command, the user can obtain such information. To obtain information About the X Processor for which an IT is being edited, the user simply omits the X Processor name. Thus, the sequence might be

```
\24,SVPROP=:\
\:ATTRIBUTES,,SVPROP
```

ATTRIBUTES DISPLAY FOR PROCESSOR GPMP — VERSION 6

ENTRY	KEYWORD	SIZE	ROW	COL	CLASS	TYPE	I/O	R/O
1	GEOCON	60	15		DE	D	I	R
2	PHYCON	60	15		DE	D	I	R
3	GSL	4	1		DE	D	I	R
4	LBMKG	4	1		DE	D	I	R
5	FTM	4	1		DE	D	I	R
6	SESCON	288	144		DE	F	I	R
7	BDATE	164	82		DE	F	I	R
8	PROCON	14	7		DE	F	I	R
9	INUEC	56	28		DE	F	I	R
10	THRUST	2	1		DE	R	I	R
11	ISP	2	1		DE	R	I	R
12	ENGID	2	1		DE	C4	I	R
13	MANID	2	1		DE	C4	I	R
14	DELUOPT	2	1		DE	C4	I	O
15	DUCOMP	6	3		DE	R	I	O
16	ANGLE	2	1		DE	R	I	O
17	ALTOPT	2	1		DE	C4	I	O
18	ALTITUDE	2	1		DE	R	I	O
19	APSOPT	2	1		DE	C4	I	O
20	HA	2	1		DE	R	I	O
21	HP	2	1		DE	R	I	O
22	NUMORB	2	1		DE	R	I	O
23	ONSFLAG	2	1		DE	C4	I	O
24	SUPROP	2	1		DE	C4	I	R
25	PROPCON	22	11		DE	F	I	O
26	CD	2	1		DE	R	I	O
27	AREA	2	1		DE	R	I	O
28	DENSMODL	0	150		DE	F	I	O
29	SCOEF	32	16		DE	F	I	O
30	SOLRFLUX	246	123		DE	F	I	O
31	GEOPOTEN	0	86		DE	F	I	O
32	DISPLAY	2	1		DE	C4	I	R
33	DSPUNITS	288	144		DE	F	I	R
34	IGUECNAM	8	1		DE	C16	I	O
35	BRUECNAM	8	1		DE	C16	I	R
36	DSPTIME1	4	2		DE	C4	I	R
37	DSPTIME2	4	2		DE	C4	I	R
38	GPMPTET	0	60	3	DE	F	O	R
39	SUMTABLE	368	4	46	DE	F	O	R

35-1. The ATTRIBUTES display for an X Processor provides information about all of its parameters.

The ATTRIBUTES command allows the user to display the attributes of all or selected parameters of a specified X Processor or document segment.

```
ATTRIBUTES-device ID,object name-class,parameter keyword,...
```

The PERFORM command causes a particular X Processor to be executed.

```
PERFORM,X Processor name,interface table name
```

The ATTRIBUTES command can also be used to obtain information about the parameters associated with a particular document/segment. This use of ATTRIBUTES is discussed further in the later section on document preparation.

```
ATTRIBUTES-device ID,object name-class,parameter keyword,...
PERFORM,X Processor name,interface table name
```

35-2. Syntax of the ATTRIBUTES and PERFORM commands.

X Processors are executed by using the PERFORM command. Ordinarily, the user will specify not only the X Processor name, but also the name of the interface table to be used for input/output control. Thus, a typical PERFORM command is

```
PERFORM,GPMP,OMS2
```

It is possible, however, to initiate execution of an X Processor using the default IT for that processor, as by the command

```
PERFORM,GPMP
```

Whenever a value is needed which is not provided in the default IT (or whatever IT is in use), the editor will automatically be invoked by the system, in order to obtain the value from the user. Thus, it is possible to successfully execute a processor without a specially prepared interface table, although in most cases this is rather tedious. This approach also has the disadvantage that the values input are not permanently stored, and must be re-entered if the processor is executed again. Like most commands, PERFORM can be used as a statement in a sequence table. It will be very common to collect a whole series of PERFORM commands in a sequence table in order to execute them in one operation.

As was just indicated, the usual mode of FDS use involves construction of interface tables before X Processor execution. During the IT editing process, a variety of commands may be used (all from the update mode, of course). Included among the commonly used commands are LIST, CHANGE, and ATTRIBUTES. The ATTRIBUTES command is helpful when the user is uncertain of the properties of a parameter and must know them in order to successfully complete the editing operation. Thus, during initial construction of the OMS2 IT, the user might encounter a prompt for the parameter PROCON and be unsure of the number of elements to be provided. Use of the ATTRIBUTES command settles this issue.

```
\8,PROCON=:\
\:AT,,PROCON
```

The user will recall that command keywords can be shortened to as few as two characters, if desired. The "AT" here is short for "ATTRIBUTES".

```
ATTRIBUTES DISPLAY FOR PROCESSOR GPMP          VERSION 6
ENTRY   KEYWORD   SIZE   ROW   COL   CLASS   TYPE   I/O   R/O
   8    PROCON      14     7          DE      F      I     R
\:MODE,I
\8,PROCON=:
```

When the editing has been completed, the GPMP processor can be executed.

```
PERFORM,GPMP,OMS2
```

Alternatively, this PERFORM command can be placed in a sequence table for later execution as part of sequence.

36 AUTOMATIC, SEMIAUTOMATIC, and BATCH Commands (Execution of Sequence Tables)

The AUTOMATIC, SEMIAUTOMATIC, and BATCH commands allow execution of sequence tables with varying levels of user interaction.

The AUTOMATIC command executes a sequence of statements which have been prestored in a designated sequence table. Each statement is executed as it is encountered in the sequence table. User interaction will occur only if the system detects incomplete data, or if an X Processor is executed which is itself interactive with the user. In the absence of these circumstances, the entire sequence table will be executed before any further user input is required. The user specifies the sequence table to be executed

```
AUTOMATIC,TRAJECT
```

The ST must be in the AWA for execution to occur. The user may specify that a portion of the ST is to be executed, by providing starting and/or ending entry numbers. Thus,

```
AUTOMATIC,TRAJECT(20-230)
```

executes entries 20 through 230 of ST TRAJECT before returning control to the user.

```
AUTOMATIC-trace level,sequence table name(span)

SEMIAUTOMATIC,sequence table name(span)

BATCH-trace level,data base,sequence table name(span),job limits
```

36-1. Syntax of the AUTOMATIC, SEMIAUTOMATIC, and BATCH commands.

By specifying a trace level, the user can cause the system to print statements before executing them. If the trace level selected is "C", only "comments" inserted in the ST via the NOTE command are printed.

```
AUTOMATIC-C,TRAJECT
```

If the trace level is "A" (for "ALL"), the system prints both NOTES and executable statements

```
AUTOMATIC-A,TRAJECT
```

The SEMIAUTOMATIC command behaves in the same way, except that it stops for user action before each statement in the sequence table. If, for example, the user executes statements 1000 through 1200 of the TRAJECT sequence table,

```
SEMIAUTOMATIC,TRAJECT(1000-1200)
```

the system will respond with

```
1000,PERFORM,GPMP,ETSEP
$1000:
```

This response indicates the statement which is about to be executed, as well as its entry number in the table. The user may cause the statement to be executed by pressing the carriage return. Alternatively, the user may:

- Skip to the next entry (by typing $)
- Specify the entry number of another statement at which execution is to begin
- Type in a statement to be executed
- Abort the ST execution (by typing %)
- Obtain a list of legal commands (?) or the syntax of a particular command (by typing, e.g., LIST?)

AUTOMATIC:

Interacts with user only when

(1) incomplete data detected (Editor is invoked)
(2) an X Processor directly interacts with user

SEMIAUTOMATIC:

As above, and before each statement.

User can skip statement, override it, or allow it to be executed

BATCH:

None. User need not even be logged on when ST is executed.

36-2. These commands differ primarily with regard to interaction with the user.

The AUTOMATIC, SEMIAUTOMATIC, and BATCH commands cause the execution of entire sequence tables. AUTOMATIC executes the entire ST without user interaction (unless incomplete data exist or an interactive processor is executed).

AUTOMATIC-trace level,sequence table name(span)

SEMIAUTOMATIC halts before the execution of each statement, allowing the user to concur or to skip or overide the statement.

SEMIAUTOMATIC,sequence table name(span)

BATCH causes the later execution of the ST in background mode, with no terminal interaction.

BATCH-trace level,data base,sequence table name(span),job limits

RESPONSE	EFFECT
Carriage Return	Statement is executed as displayed.
$	System skips to next statement in ST.
Entry number (e.g., 1210)	System skips to indicated statement in ST.
Any legal statement	System executed this statement instead of the one it displayed.
%	ST execution is aborted and control is returned to the FDS Executive.
?	System displays a list of legal statements.
Command? (e.g., LIST?)	System displays the correct syntax of the specified statement.

36-3. During SEMIAUTOMATIC execution, the user has several options available each time the system displays the "next statement".

The BATCH command causes FDS to execute a sequence table later, as a "batch", or background, job. This allows more efficient system operation when the user does not require instant turnaround, and also allows the user to leave or do other work while the job is executing. It also has the advantage that it produces a hardcopy (lineprinter) record of the entire execution. Since no AWA will exist for this job at the time execution is begun, the user must specify the name of a PDB or MDB. Thus, the command

BATCH,EXPL-PD,TRAJECT

indicates that FDS is to use the PDB called EXPL as if it were the AWA, obtaining the TRAJECT file from that PDB, as well as all required interface tables, data elements, etc. When the system has executed this command (i.e., submitted the job for later execution), it will return a job name to the user, who may then log off or do other FDS work. Later, when FDS executes the job, it will work from the EXPL PDB as it exists then, so it is important that it be left intact until the job has finished executing.

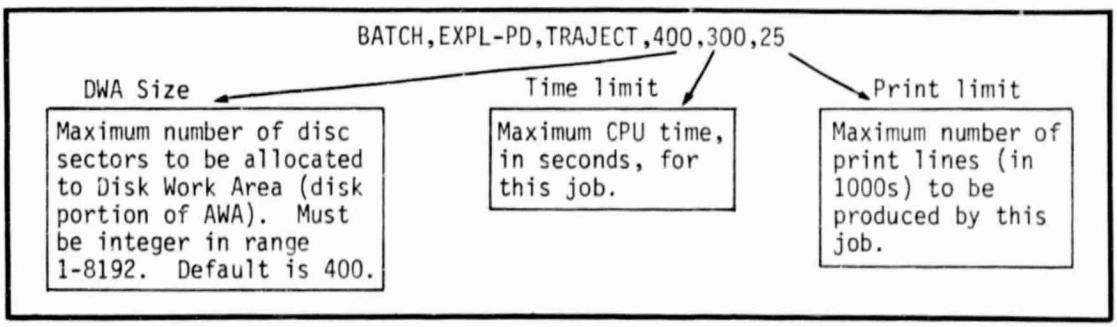
The BATCH command can also specify a trace level, execute only a segment of a sequence table, and specify job limits. Thus, the command

BATCH-A,FLIGHT-MD,BUILDPLN(10-200)

directs FDS to execute lines 10-200 of the BUILDPLN sequence table from an MDB called FLIGHT, and to list each statement before executing it. Job limits can be added in order to control the disk space, CPU time, and number of print lines allowed. The default values of these parameters should be satisfactory for most batch jobs, but the user may override them when necessary. For example, the command

BATCH,HISPDB-PD-AY,OFT2,800,240,25

sets limits of 800 disk sectors, 240 cpu seconds, and 25,000 print lines.



36-4. The "job limits" portion of the BATCH command has three components.

37 ALLOCATE, ASSIGN Commands (Data Element Creation and Initialization)

The ALLOCATE command is used to create a data element in the AWA with user-specified properties, while ASSIGN is used to place information into an already allocated DE.

```
ALLOCATE,data element name(dimension,dimension)-data type,...

ASSIGN,data element name(expression,expression)=expression 'range '...;
```

37-1. Syntax of the ALLOCATE and ASSIGN commands.

The ALLOCATE command creates one or more new data elements in the AWA. In this command, the user specifies the name of the new DE, which must not conflict with any existing DE names. The user also specifies the data type which the DE will contain. For example, the command

```
ALLOCATE,NUMVAL-I
```

creates a DE called "NUMVAL" which will contain a single integer value. The data types available are integer (I), real (R), double precision (D), character (Cn), free (F), time (T), or mixed (Mn, where n = format number). ALLOCATE can also specify the dimensions of a DE which is to be used as an array. These arrays may have one or two dimensions, and are specified by giving the upper limit of each subscript, as might be done, for example, in a FORTRAN DIMENSION statement. Thus,

```
ALLOCATE,ONEDIMEN(16)-R,TWODIMEN(21,4)-D
```

creates two new data elements. The DE called ONEDIMEN is to contain an array of 16 real values, which will occupy 64 bytes of storage (each real variable takes 4 bytes of storage). Values in this array will be referred to by name and subscript, so that ONEDIMEN(3) is a reference to the third value in this array. TWODIMEN contains 84 double precision values (672 bytes) and is referenced with two subscripts. For example, TWODIMEN(6,2) refers to the second value in the sixth row of the 21 x 4 array. Data elements may not be allocated for more than 4095 bytes, or an error message will result.

DATA TYPE	BYTES PER VALUE
integer (I)	2
real (R)	4
double precision (D)	8
character (Cn)	4,8,16,32,or 64
free (F)	words allocated (4 bytes each)
time (T)	8
mixed (M)	combinations of above types

37-2. The data type of a DE affects its storage requirements

The ASSIGN command computes values and stores them in a DE. Functionally, ASSIGN is very similar to the assignment statement found in FORTRAN or other high-level programming languages. Thus the command

```
ASSIGN,NUMVAL=10;
```

simply places the value 10 in the (single-valued) data element called NUMVAL. Expressions for the value can be arbitrarily complex, involving addition, subtraction, multiplication, division and/or exponentation. They may include a variety of mathematical functions (see Appendix A) and nested parenthetical expressions. Expressions are evaluated in the conventional way (as in FORTRAN, for example), so that the command

```
ASSIGN,NUMVAL=4*2**3-6/2+3;
```

is evaluated as if it had been written

```
ASSIGN,NUMVAL=(4*(2**3))-(6/2)+3;
```

The ALLOCATE command is used to create one or more data elements in the AWA. The user must specify the name and type of the data element and may specify its dimensionality.

ALLOCATE,data element name(dimension,dimension)-data type,...

The ASSIGN command computes values and stores them in an existing AWA-resident DE.

ASSIGN,data element name(expression,expression)=expression 'range '...;

involves multiplying an integer (2) by a data element value (using whatever data type it has), multiplying the result by a real constant, and converting the result back to an integer (NUMVAL was ALLOCATEd, above, as an integer). No conversion is applied to free data, however. Character strings may also appear as values, as in the command

```
ASSIGN,HOLDCHAR="SPACE TRANSPORTATION SYSTEM";
```

but arithmetic operations are then not allowed. The ASSIGN command must be terminated with a semicolon.

The data element name, to which the value is assigned, can also be subscripted. Subscripts can be any valid numerical expressions, as defined above. Thus,

```
ASSIGN,ONEDIMEN(2**2-14/7)=12;
```

is valid, and is equivalent to

```
ASSIGN,ONEDIMEN(2)=12;
```

Naturally, references to two-dimensional arrays require two subscript expressions:

```
ASSIGN,TWODIMEN(ONEDIMEN(1),6-NUMVAL)=26.183;
```

ASSIGN can also be used iteratively, to assign values to more than one element of an array. This is done in a manner similar to the ordinary DO statement in FORTRAN. For example, the command

```
ASSIGN,TARGET(I)=SOURCE(I) 'I=1,10;
```

copies the values of an entire ten-element array (SOURCE) into another array (TARGET). The range specification is indicated by an apostrophe, and contains the following information:

'index=initial value,terminal value,increment

where the increment is optional. The "index" is a variable name. The variable takes integer values as indicated by the remaining information. For the first iteration, the index is equal to the initial value. On each successive iteration, the index value is increased by the increment until it exceeds the terminal value. Thus

```
'IVAL=1,10,3
```

causes IVAL to take on the values 1, 4, 7, and 10. The next value, 13, is past the terminal value, so the fifth iteration does not occur. If the increment is not specified, a value of 1 is used (-1, if terminal value is smaller than initial value). More than one range specification can be used, to a maximum of four. Thus,

```
ASSIGN,PRODUCT(I,J)=I*J
CONTINUE:  'I=1,6 'J=1,4;
```

assigns values to all 24 elements of a 6 x 4 array. In each case, the value assigned is equal to the product of the row and column numbers by which it is indexed.

```
Evaluate and store a root of a quadratic equation
     ASSIGN,R1=(-B+SQRT(B**2-4*A*C))/(2*A);
The range field can be used to set values in a data element.
     ASSIGN,VECTOR2(NUMBER) = COUNT(NUMBER) 'NUMBER= 1,10;
Compute and store the scalar product of two vectors
     ASSIGN,DOT=0;
     ASSIGN,DOT = DOT + A(IND) * B(IND) 'IND = 1,3;
Character data can be stored in DEs created for character data
     ALLOCATE,TITLE-C32
     ASSIGN,TITLE = "ROOT1 AND SCALAR PRODUCT";
```

37-3. ASSIGN can be used for a variety of purposes.

38 IF Statement (Condition Testing in Sequence Tables)

The IF...ELSE...ENDIF (or IF...ENDIF) block allows selective execution of sequence table statements.

The IF statement is used in sequence tables to allow conditional execution of one or more statements. In its simplest form, the IF block consists of an IF statement, one or more other statements, and ENDIF.

```
IF,INUM>6;
ASSIGN,INUM=6;
ENDIF
```

For instance, the above sequence sets the value of data element INUM to 6 if and only if it started with a value greater than 6. Logically, the condition specified in the IF statement is tested to determine whether it is true or false. If it is true, the statement(s) following the IF statement are executed; if false, they are skipped until ENDIF (or ELSE) is found. If an ELSE clause is present,

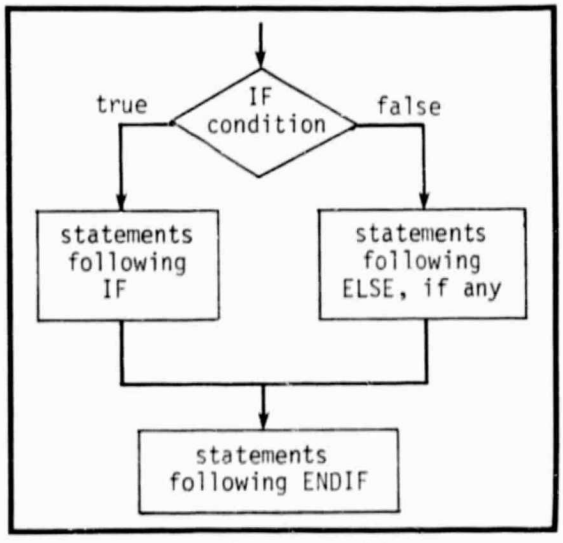
```
IF,INUM>6;
ASSIGN,INUM=6;
ELSE
ASSIGN,INUM=0;
ENDIF
```

then the statements following ELSE are executed only if the condition is false, and are skipped if it is true. Thus, either of two blocks of statements can be selected, depending on the truth value of the condition. In the above instance, INUM is set to 6 if it started out greater than 6, and is set to 0 if it started out less than or equal to 6. Any number of statements can occur after IF and after ELSE. IF, ELSE, and ENDIF can appear only in sequence tables; they cannot be entered as directives for immediate execution by the FDS Executive. Incidentally, the IF statement must be terminated with a semicolon.

```
IF,condition 'range '...;
(statements)
ENDIF

IF,condition 'range '...;
(statements)
ELSE
(statements)
ENDIF
```

38-1. The IF statement can be used with or without an ELSE-clause.



38-2. The function of IF blocks is illustrated by a flowchart.

The "condition" portion of an IF statement consists of two mathematical expressions and a relational operator. The IF statement amounts to a construct of the form

expression relation expression.

The expressions can take any of the forms discussed in connection with the ASSIGN statement. They can include addition, subtraction, etc., as well as parentheses and mathematical functions. They may refer to data elements, with or without subscripts, as well as numerical constants. The relations include equal (=), not equal (<> or ><), less than (<), greater than (>), less than or equal (<= or =<), and greater than or equal (>= or =>). Thus, very complex tests are possible, although it will usually be advantageous to keep things simple.

The IF...ELSE...ENDIF (or IF...ENDIF) block allows conditional execution of statements in a sequence table.

```
IF,condition 'range '...;
(statements)
ELSE
(statements)
ENDIF
```

IF blocks may be nested in a hierarchical fashion. Thus, the sequence

```
IF,I=1;
IF,J<10;
LIST,XYZ-ST
ENDIF
ELSE
LIST,WXY-ST
ENDIF
```

is allowed. In this sequence, the inside IF block is executed only if I equals 1. If I is not equal to 1, the ELSE clause is executed (i.e., the statement "LIST,WXY-ST" is executed). Notice that, if I is 1 and J is 10 or greater, nothing is LISTed. Interpretation of such a construct may be easier if it is written in indented form to show its hierarchical structure.

```
IF,I=1;
   IF,J<10;
      LIST,XYZ-ST
      ENDIF
ELSE
   LIST,WXY-ST
   ENDIF
```

Symbol	Operation
<	less than
>	greater than
<=	less than or equal
=<	" " " "
>=	greater than or equal
=>	" " " "
=	equal
><	not equal
<>	" "

38-3. Several relational operators can be used in the "condition" portion of IF statements.

An IF statement can also have a "range". For example, the statement

```
IF,MATRIX(I)>0 'I=1,10;
```

could be used to test all values of a 10-element array. The condition is true only if all individual tests are true. Thus, in the above case, all 10 elements of MATRIX must be greater than zero or the condition is false. Up to four range specifications can be used. Their form is the same as that discussed in connection with the ASSIGN statement.

CONDITIONAL EXPRESSION*	EQUIVALENT ARITHMETIC EXPRESSION (assume I=2,J=2,K=3)	TRUTH VALUE
I = J	2 = 2	TRUE
I = K	2 = 3	FALSE
I < J	2 < 2	FALSE
I < K	2 < 3	TRUE
I <= J	2 <= 2	TRUE
I > J	2 > 2	FALSE
K > I	3 > 2	TRUE
I >= J	2 >= 2	TRUE
I <> J	2 <> 2	FALSE
I <> K	2 <> 3	TRUE
I*J >= K	4 >= 3	TRUE
I**J = (K-1)**J	4 = 1	FALSE

38-4. The truth value of a condition is determined by computing the numerical values of the expressions and applying the relational operator.

39 DO LOOP Statement (Iteration In Sequence Tables)

The DO LOOP block is used for conditional looping in sequence tables.

The DO LOOP...LOOP END block allows iterative execution of statements as a function of a specified condition. Thus, the sequence

```
ASSIGN,I=1;
DO LOOP,WHILE,I<=10;
ASSIGN,ARRAY(I)=I;
ASSIGN,I=I+1;
LOOP END
```

causes the last two ASSIGN statements to be executed 10 times, with the data element I taking on the values 1, 2, 3, ... 10. The loop has the effect of setting each element of ARRAY to its index position (ARRAY(1) = 1, ARRAY(2) = 2, etc.). This is an example of a WHILE loop, in which the loop is repeatedly executed while the specified condition is true. The test is performed each time the loop is entered. When it is found to be false, the loop is exited, and those statements which follow LOOP END are executed. DO LOOP (and LOOP END) statements can be used only in sequence tables. DO LOOP statements must end in a semicolon.

```
DO LOOP,WHILE,condition 'range '...;
(statements)
LOOP END

DO LOOP,UNTIL,condition 'range '...;
(statements)
LOOP END
```

39-1. The DO LOOP statement has both WHILE and UNTIL forms.

In the UNTIL form of DO LOOP, the test is performed at the end of the loop. Compare the sequence

```
ASSIGN,I=1;
DO LOOP,WHILE,I>=10
ASSIGN,I=9999;
LOOP END
```

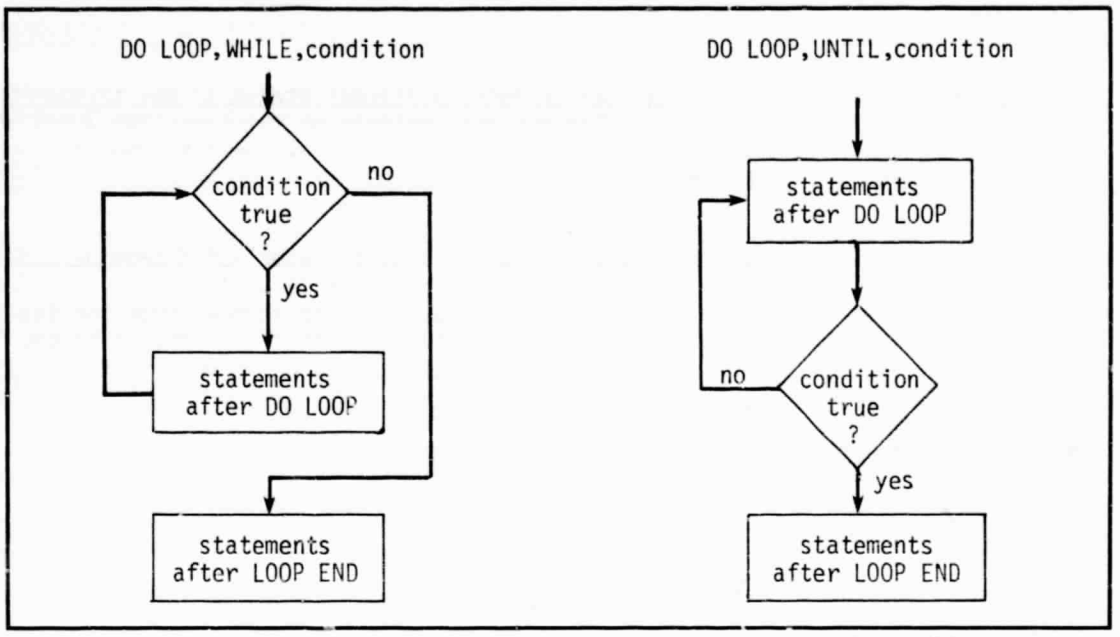
with the sequence

```
ASSIGN,I=1;
DO LOOP,UNTIL,I>=10
ASSIGN,I=9999;
LOOP END
```

When the WHILE sequence is executed, I is given as value of 1, which is not changed by the DO LOOP. This is because the condition I>=10 is false and the statement within the DO LOOP block is not executed. On the other hand, the UNTIL loop is executed exactly once. Since no test is performed until after the statement within the loop is executed, I takes on the value 9999. When the test is then applied, it is found to be true, which causes the loop to be exited. As the words suggest, WHILE causes iteration as long as the condition remains true, and UNTIL causes iteration until the condition becomes true.

The DO LOOP block allows conditional looping within a sequence table. A collection of sequence table statements can be repeatedly executed WHILE specified conditions are met or UNTIL specified termination conditions are satisfied.

```
DO LOOP,WHILE,condition 'range '...;
or DO LOOP,UNTIL,condition 'range '...;
(statements)
LOOP END
```



39-2. WHILE and UNTIL loops differ in the time at which test is applied.

As in the IF statement, the condition in a DO LOOP statement can involve a "range". To be considered true, the condition must be true across all values of the index variable. For example, the loop

```
DO LOOP,WHILE,ARRAY(I)>0 'I=1,5;
(statements)
LOOP END
```

will be skipped, if any of the first five elements of ARRAY is nonpositive. Up to four range specifications can be used.

DO LOOP blocks can be nested with other DO LOOP blocks or with IF blocks. Thus, a sequence such as

```
ASSIGN,I=1;
DO LOOP,WHILE,I<=10;
IF,B(I)>A(I);
ASSIGN,A(I)=B(I);
ENDIF
ASSIGN,I=I+1;
LOOP END
```

is possible. This sequence causes the first 10 elements of A to take on a value which is the greater of: (1) their initial value, and (2) the corresponding value in array B. When blocks are nested in this way, it is important to recognize that the inside block must be closed first. Thus, the following "overlapping" structure is illegal.

```
DO LOOP,WHILE,I<6;
IF,J=0;
(statements)
LOOP END
ENDIF
```

40 BREAK, DLOG, FORMAT, MESSAGE, NEWS Commands (Miscellaneous FDS Commands)

The commands BREAK, DLOG, FORMAT, MESSAGE, and NEWS allow the user to obtain system status, transaction log, mixed data format displays, and current news information, and to communicate with other users.

The BREAK Command causes FDS to display the current functional status of all active FDS tasks associated with the current user. The current contents of the execution stack are displayed to the user. The appropriate prompt symbol, task name, and the names of any tables being processed are displayed.

The DLOG command allows the user to display all or part of the user's transaction log file. The transaction log is a "circular" file which has a predetermined size. As new transactions are added, the oldest ones are thrown away in order to make room for the new information. Each FDS prompt, each user input, and each error message, is placed in this file, along with date and time tags. The user can display all transactions for the current day on the terminal

```
DLOG
```

or on the system line printer

```
DLOG-P
```

or can specify starting and/or ending dates or times. For example, the command

```
DLOG,12/18/83,14/23/00
```

causes all stored transactions, which occurred on 18 December 1983, after 2:23 P.M., to be displayed. Date fields take the form month/day/year. The current day is assumed, if date is omitted but time is given, as in

```
DLOG,,14/23/00
```

Time fields take the form hour/minute/second. If start time is omitted, but other date or time information is given, time 00/0 /00 (midnight) is assumed. If end time is omitted, 23/59/59 is assumed. If a start date is given, but no end date is given, the start date is assumed to be the end date. Thus, the command

```
DLOG,12/17/83,,,14/23/00
```

displays all stored transactions from the start of the day on 17 December 1983 through 2:23 P.M. on the same day.

```
12/18/83  08/04/24  * INFO  *EX*XMINIT01* 53*
        FDS SIGNON COMPLETED,FDS VERSION IS 6C
12/18/83  08/04/24  %  :
12/18/83  08/04/45  GET,EXPL-PD
12/18/83  08/04/46  %  :
12/18/83  08/04/59  TOC
12/18/83  08/05/01  %  :
12/18/83  08/05/27  EDIT,,OMS2-IT,GPMP
```

40-1. The transaction log contains a time-stamped record of recent FDS prompts and user inputs.

The FORMAT command displays current definitions of all, or selected, mixed data formats. The mixed data type is used when elements of more than one type are contained in a single data element or DRDE. While data types may be combined, they must be combined according to an existing format definition. These definitions are numbered, and may be displayed using the FORMAT command. Thus,

```
FORMAT,M31,M33
```

displays the contents of format definitions 31 and 33, while

```
FORMAT-P
```

displays all current format definitions on the system line printer.

The BREAK command displays the current functional status of all active FDS tasks associated with the current user.

```
BREAK
```

The DLOG command displays all or part of the user's transaction log file.

```
DLOG-device ID,start date,start time,end date,end time
```

The FORMAT command displays current mixed data formats.

```
FORMAT-device ID,format number,...
```

The MESSAGE command transmits a message to one or more other users.

```
MESSAGE,access code(s),"text"
```

The NEWS command displays system status information which has been placed in a NEWS file by system support personnel.

```
NEWS-device ID,start date
```

The MESSAGE command allows the user to transmit a message to any other FDS user. The message will be displayed for the recipient(s) with the date and time of transmission and the originating user's access code. If a recipient is currently logged on, the message will be displayed on that user's terminal just prior to the next Executive level (*$*) prompt. If the recipient is not logged on, the message is retained and automatically presented when the user next logs on. Once a retained message has been displayed, it is purged from the system.

```
BREAK
DLOG-device ID,start date,start time,end date,end time
FORMAT-device ID,format number,...
MESSAGE-access code(s),"text"
NEWS-device ID,start date
```

40-2. Syntax of miscellaneous FDS commands.

The MESSAGE command specifies the message text, and may specify the access code(s) of one or more recipients. For instance, the command

```
MESSAGE,AV,"MEETING HAS BEEN POSTPONED TO 3:30"
```

transmits a message only to user AV. If no access code is given,

```
MESSAGE,,"SYSTEM WILL BE DOWN 3:30-4:00 P.M."
```

it is transmitted to all users. The user should keep in mind the message retention feature. The above message is clearly unsatisfactory. A user who next logs on a week later may not recognize this message as inapplicable at that time. Multiple recipient access codes are allowable, as in

```
MESSAGE,AW,CW,"HAS ANYONE UPDATED THE
CONTINUE: OMS-2 INTERFACE TABLE YET?"
```

This command takes advantage of the multiple line facility available in all commands. Note, however, that the message will be displayed in a single line

```
MESSAGE FROM CY,12/18/83 02/16/27
HAS ANYONE UPDATED THE OMS-2 INTERFACE TABLE YET?
```

unless Control-C is used to break it into multiple lines. The message text is limited to 122 characters, including Control-C characters.

The NEWS command displays system status information which has been placed in a NEWS file by system support personnel. This facility is used primarily to convey information about the system itself, including updates to system software and to Master Data Bases. If no device ID is specified,

```
NEWS
```

the information is displayed on the user's terminal. If the line printer device code is specified,

```
NEWS-P
```

the news is printed on the system printer. If no date is specified (as in the above commands), the entire news file is displayed. To obtain only news entered on or after a specified date, that date is added to the command in the form month/day/year.

```
NEWS,12/18/83
```

41 AUTOMATED DOCUMENT PRODUCTION

FDS supports automated production of standard documentation.

A major function of FDS is its support of automated document production. Briefly, FDS accomplishes this by substituting computed and user-provided information for "holes", or placeholders, in a predefined text document. The overall document production process is fairly complex, involving five groups of personnel, three computers, and a large number of procedural steps. From the user's point of view, though, the process is fairly simple and automatic.

In automated document production, the user's job is to provide the information which changes from mission to mission. Each standard document (e.g., CFP) has a large amount of information which remains the same across all, or many, flights. Only the changeable information needs to be provided in order to produce documentation which fully describes a unique mission. Support personnel have created a number of standardized document segments, and have left spaces ("holes") to show where user-provided information goes. The information already provided by the support personnel includes format specifications, units of measure (conversion is automatic), and other data required to fill each "hole". The user must only indicate to FDS the values, or where the values are to be found, so that FDS can automatically fill in all the "holes" in a document/segment.

Although the user's part in document production is relatively simple, a clear understanding of the process requires an awareness of several components. For each DOCUMENT/SEGMENT, there is a FORM, which consists of the basic text (and tables, etc.) of the segment, with placeholders showing where values are to be provided by FDS. These forms are provided by system support personnel. For each document/segment, there is also a DATA DESCRIPTOR TABLE (DDT) provided by support personnel. This table indicates the name of each missing value or set of values in the Form, and provides information about format, units of measure, etc. Although this information is actually stored in FDS Linkage Tables, it is convenient to think of it as a separate entity; it is not under user control, as is the rest of the Linkage Table information. The DOCUMENT PROCESSOR uses this DDT information, and information provided by the user, to construct a finished document. The Document Processor is an X Processor, and is executed in the same way as any other X Processor, using a PERFORM command which specifies an Interface Table.

The components which the user must provide are the Linkage Table and the Interface Table. The LINKAGE TABLE (LT) is the user's way of identifying, for the system, the values which are to be placed in the "holes" in the Form. Because the Document Processor (DOC) is an X Processor, it obtains its basic information from an Interface Table. In this case, the IT identifies the name of the Document/Segment, name of the Linkage Table, and other information necessary for the DOC to proceed.

FDS supports automated production of standard documents with minimal effort by the user. To produce a document, the user must provide the values (or their locations) which are to be placed in the "holes" in a standard Form. This process utilizes Forms and Data Descriptor Tables (both provided by support personnel), a Documentation Processor (DOC-XP), and Linkage Tables and Interface Tables (both provided by the user).

Form:

Text, tables, etc., with "holes" to show where FDS-supplied values are to be placed. These Forms are provided by support personnel.

Data Descriptor Table (this information is actually stored as part of the default Linkage Table):

A description of the data items which are to be placed in the "holes". This table provides a name for each data item, information about its format, and unit of measure (where appropriate). The order of the entries in the DDT corresponds to that of the "holes" in the form. DDTs are provided by support personnel.

Linkage Table (user-modifiable portion):

A table which indicates the values which are to be placed in the "holes", or the names of data elements or DRDEs in which the values are stored. This table is the user's means of causing appropriate values to appear in the automatically produced document.

Interface Table:

An ordinary interface table (for the DOC X Processor) which tells the Document Processor the name of the document/segment, linkage table, etc., needed to prepare a finished document/segment.

Document Processor:

An X-Processor (called DOC) which performs the document preparation. DOC obtains the Form and uses information in the DDT and LT to fill in all the "holes". The resulting document/segment can be displayed for user review, and can be deleted or stored for actual production.

41-1. Production of a finished document/segment requires several components.

42 AN EXAMPLE OF AUTOMATED DOCUMENT PRODUCTION

The user controls the behavior of the Document Processor by making appropriate entries in a Linkage Table and an Interface Table.

Preparation of a Linkage Table is the main user activity involved in document production. In almost all respects, this is just like preparing an interface table. Suppose, for example, that the user wishes to prepare a linkage table called EXAMPLE, starting with the default Linkage Table for a document called CFP1, segment LIFT. The user would type in the command

```
EDIT,,EXAMPLE-LT,CFP1LIFT
```

Just as in interface table editing, the editor starts in the incomplete mode unless the default LT is complete. In the LT, the parameters are the names of the items of information which are to be substituted for placeholders ("holes") in the form. As in an IT, the user can specify the actual value to be used, or can specify the name of a DE where the information can be found. Thus, in response to an editor prompt, the user could type in an actual value

```
1,LSITET=&1
<&1:"KSC"
```

or the name of a data element which contains the value

```
<&1:SITE
```

When arrays are required, the whole set of values can be typed in, or the name of a DE containing the whole set can be provided. The user can also indicate a particular position in an array. Assume, for example, the lift-off time is ordinarily found in element 7 of the first row of the flight state vector. Then, the user might respond

```
2,TLO=&1
<&1:RESULT(1,7)
```

In order to point the document processor to element (1,7) of the RESULT DE. If TLO required more than one value, the required number of values would be taken from RESULT, starting in location (1,7).

The user must also provide an Interface Table, which DOC uses to find all the information it needs, to write to the correct file, etc. Thus, the user might construct an IT called EXAMIT, using ordinary procedures for IT editing. The user must state the name of the document (here it is assumed to be CFP1), the segment name (LIFT), Linkage Table name (EXAMPLE), output test file name (OUTTEXT) and other information. A portion of this dialogue might be:

```
EDIT,EXAMIT-IT,DOC
1,DOCUMENT=&1
\&1:"CFP1"
2,SEGMENT=&1
\&1:"LIFT"
3,LTABLE=&1
\&1:EXAMPLE
4,FILENAME@&1
\&1:OUTTEXT
```

For more detail about the use of DOC, the user should consult the X Processor User Reference Manual.

Once the LT and IT have been constructed, the user simply executes DOC. The appropriate command is

```
PERFORM,DOC,EXAMIT
```

DOC performs the document construction, and gives the user an opportunity to screen the results. When the user is satisfied, the output text can be stored for transmission to the Xerox Word Processing System, where final output will be accomplished.

An example illustrates the information used by the Document Processor to produce documents in an automated way. For further information, the user is referred to the X Processor User Reference Manual section dealing with the DOC processor.

Form (assume this document/segment is called CFP1/LIFT):

Lift-off from [ ] occurs at [ ] seconds after solid rocket booster (SRB) ignition (MET = [ ] seconds) after which a vertical rise to tower clearance phase is initiated, ending at a relative velocity of [ ] fps (MET = [ ] seconds).

Data Descriptor Table (for document/segment CFP1/LIFT):
(This information is actually stored in the Linkage Table, mostly in the "extended prompt" records. See X Processor User Reference Manual for more information.)

name	type	format	units	prompt text
LSITET	C4	A3	-	LAUNCH SITE
TLO	R	T0:0:0:1	-	TIME OF LIFTOFF(GET)
TL01	R	T0:0:0:3.1	-	TIME OF LIFTOFF(GET)
RELVEL	R	I3	FT/S	RELATIVE VELOCITY
TCLNCE	R	T0:0:0:3.1	-	TIME OF TOWER CLEARANCE

Linkage Table (assume this is called EXAMPLE-LT):

```
1,LSITET="KSC"
2,TLO=RESULT(1,1)
3,TL01=RESULT(1,2)
4,RELVEL=RESULT(7,3)
5,TCLNCE=RESULT(1,3)
```

Interface Table (assume this is called EXAMIT-IT)

```
1,DOCUMENT="CFP1"
2,SEGMENT="LIFT"
3,LTABLE=EXAMPLE
4,FILENAME=OUTTEXT
```

Date Element (called RESULT. This is just an example):

Assume that RESULT(1,7)=3.0
Assume that RESULT(2,7)=3.1
Assume that RESULT(3,9)=20.0
Assume that RESULT(3,7)=3.8

Finished text (stored by DOC in OUTTEXT-DF):

Lift-off from KSC occurs at 3 seconds after solid rocket booter (SRB) ignition (GET=3.1 seconds) after which a vertical rise to tower clearance phase is initiated, ending at a relative velocity of 20 fps (GET=3.8 seconds).

42-1. An example shows how the various components are used by the Document Processor to produce the finished document.

APPENDIX A -- MATHEMATICAL FUNCTIONS

In contexts in which mathematical functions may be used (e.g., ASSIGN command), any of the functions listed below are available. The user should be aware of a peculiarity of the operating system. When a function error occurs (e.g., ATAN(0,0)), no interrupt or error message to the user terminal occurs. The system returns a default function value and writes an error message only to the system console. This should be kept in mind when debugging operations involving functions.

Function Name	Result Type	Argument(s) Type	Number of Arguments	Operation
ABS	R	R	1	Absolute value
ACOS	R	R	1	Inverse cosine
AINT	R	R	1	Truncate fraction
ALOG	R	R	1	Natural logarithm
ALOG10	R	R	1	Base ten logarithm
AMOD	R	R	2	Modulo
ASIN	R	R	1	Inverse sine
ATAN	R	R	1	Inverse tangent
ATAN2	R	R	2	Inverse tangent
COS	R	R	1	Cosine
DABS	D	D	1	Absolute value
DATAN	D	D	1	Inverse tangent
DATAN2	D	D	2	Inverse tangent
DBLE	D	R	1	Convert to double
DCOS	D	D	1	Cosine
DEXP	D	D	1	e ** x
DINT	D	D	1	e ** x
DLOG	D	D	1	Natural logarithm
DLOG10	D	D	1	Base ten logarithm
DMOD	D	D	2	Modulo
DSIGN	D	D	2	Apply sign
DSIN	D	D	1	Sine
DSQRT	D	D	1	Square root
DTAN	D	D	1	Tangent
DTANH	D	D	1	Hyperbolic tangent
EXP	R	R	1	e ** x
FLOAT	R	I	1	Convert to real
IABS	I	I	1	Absolute value
IDINT	I	D	1	Convert to integer
IFIX	I	R	1	Convert to integer
ISIGN	I	I	2	Apply sign
MOD	I	I	2	Modulo
SIGN	R	R	2	Apply sign
SIN	R	R	1	Sine
SNGL	R	D	1	Convert to real
SQRT	R	R	1	Square root
TAN	R	R	1	Tangent
TANH	R	R	1	Hyperbolic tangent

APPENDIX B -- SIZES AND DIMENSIONS

Item	Limit
Abstract length	256 bytes = 256 characters
AWA size	32768 bytes = 8192 words
Data element column length	1024 components
Data element size	4096 bytes = 1024 words
DWA size	8192 sectors (2 Mbytes)
Entry number in table	range 1-32767
Execution stack levels	8
Interface/Linkage/Sequence table size	4096 bytes = 1024 words
MDB/PDB size	32767 sectors (8 Mbytes)
MESSAGE command text	122 characters
Messages queued per user	assignable by user
Nested repeat fields	4 levels of nesting
NOTE command text	1600 characters
PDBs per user	assignable by user (also limited by disk space)
Sequence Table entries	226
Subscript of data element	range 1-2047
TOC entries (AWA or data base)	630
X batch jobs per user	assignable by user
X batch jobs queued (all users combined)	50

APPENDIX C -- GLOSSARY

abstract -- text associated with a given file or processor which contains description of the file contents or processor function. The user is responsible for creating and updating abstracts for files in PDBs.

access code -- a two-character code by which a user logs on, and by which ownership of PDBs is indicated.

Active Work Area (AWA) -- an area of memory and disk storage utilized as working space for the construction and manipulation of data. An AWA is automatically allocated to a user at sign-on and purged from the system at sign-off.

automatic (execution mode) -- a mode in which FDS interactively executes the statements in a Sequence Table without requiring user approval at each step.

batch (execution mode) -- a mode in which FDS executes the commands in a Sequence Table at some later time, as the system's processing load allows.

character (data type) -- a data type consisting of text strings which occupy 4, 8, 16, 32, or 64 bytes (characters) of storage.

command -- a user-specified instruction to FDS. Commands which are executed immediately are called directives. Commands which are stored in sequence tables for later execution are called statements.

data base -- a set of files stored on disk which are relatively permanent, such as personal data bases or master data bases.

Data Element (DE) -- a collection of related processor input and output variables grouped together into a single file and stored under a user-defined name. Data elements can be stored in the AWA, a PDB, or an MDB.

device ID -- a code which can be added to many commands to specify the device on which generated output is to be displayed. The only allowable device code is (-P), which indicates the system printer.

directive -- see command.

Disk Resident Data Element (DRDE) -- A "data element" which is potentially too large for the memory-resident portion of the AWA, and which is therefore declared to be disk-resident.

disk work area (DWA) -- the disk-resident portion of the AWA.

document processor -- A special X Processor whose function is to produce documents based on information computed in designing a flight.

document/segment -- A portion of a standard document which can be produced using FDS. Each such portion is specified by giving its document name and segment name.

editor -- A portion of the Executive which allows the user to construct and modify tables.

Executive -- that part of FDS which communicates with the user and through which the user manages files, invokes the editor and initiates processor activity.

file -- an organized, named collection of information.

Interface Table (IT) -- A table which contains input/output information for use by an X Processor.

Linkage Table (LT) -- A table which contains information on the values which are to be placed in an automatically produced document.

Master Data Base (MDB) -- collection of permanent files which provide a source of data to all FDS users. The MDB supplies basic data on a read-only basis.

Personal Data Base (PDB) -- A collection of files "permanently" stored on disk by a user.

remote job entry -- Submission of a job to a computer, from a remote location, for subsequent processing and return of output to the originating location.

semi-automatic (execution mode) -- a mode in which FDS interactively executes the statements in a Sequence Table, stopping at each step for user approval or other action.

Sequence Table (ST) -- a group of FDS commands stored for later execution.

sign-off -- process by which communication is ended with FDS, resulting in the AWA data being purged.

sign-on -- process by which communication is established with FDS, resulting in the allocation of the AWA.

statement -- see command.

table -- a file of one of the following types: interface table, linkage table, sequence table, Y-data table.

Table-Of-Contents (TOC) -- a directory of the data contained in the AWA or in various data bases. It includes the names and attributes of all data elements and tables.

X Job -- The process, events, and output associated with the execution of a Sequence Table in the batch mode.

X Processor -- A computational support routine available to the FDS user for either interactive or batch execution. X Processors reside in FDS and perform simulations at an appropriate level of detail for preliminary planning and development of a Conceptual Flight Profile.

Y Processor -- a computational support routine which resides on a separate computer from FDS and is available for execution via remote job entry. Y Processors perform highly detailed simulations compatible with development of an Operational Flight Profile.

APPENDIX D -- TIME DATA TYPE

1. Base Date:

- User specified as year, month, and day of month
- Defined to be zero hours, zero minutes, zero seconds, Greenwich zone time on the user-specified year, month, day of month
- Base date is the instant in time at which the FDS inertial reference axis systems (MEE, TEE, and TEG) are defined (i.e, the Precession matrix, Nutation matrix, and initial right ascension of Greenwich are computed)
- User specifies the Base Date for a session by executing the BASTM processor

2. Time References: In FDS-2 there are four time references available for user input and display of event times. They are defined as fllows:

IET = Internal Elapsed Time - it is an event time which is measured from the user defined base date. All internal FDS time tags (e.g., position/velocity state vector or attitude event time tags) are stored in this measure. The time is expressed in days, hours, minutes, and seconds.

MET = Mission Elapsed Time - it is an event time which is measured from a user defined reference time. The time is expressed in days, hours, minutes, and seconds.

PET = Phase Elapsed Time - it is an event time which is measured from a second user defined reference time. The time is expressed in days, hours, minutes, and seconds.

GMT = Greenwich Mean Time - it is an event time which is measured from the beginning of the year of base date. The time is expressed in day of year and time of day. The day of year is an integral day count from the beginning of the base date year. The day count is negative in the years prior to base date year and continues to increment positively in the years after base date year. Day of year does not recycle to one at the end of base date year when time tag is beyond that year.

Time of day is expressed as hours, minutes, and seconds in Greenwich zone time. For example, if the base date year is 1980 and the date to be expressed in GMT is Dec. 1, 1980, noon GMT, the GMT is 366:12:0:0. If the date is Jan. 1, 1981, noon GMT, the GMT is 367:12:0:0. If the date is Dec. 31, 1979, noon GMT, the GMT is -1:12:0:0. There is no zero day of year.

3. Specification of MET and PET Reference Times:

- The user must specify the MET and PET reference times (i.e., the MET = 0 and PET = 0 times) in the IET reference.
- The user can specify and change MET and PET reference times independently of each other utilizing the BASTM processor.

4. Time Type for Interface Table Prompt Parameters and Processor Prompts: The letter T designating the time type is used to identify it in the Processor Interface Table and Processor Prompt Table. The input user syntax for time is ± d.d:h.h:m.m:s.s where the sign applies to the total time value. Each number is any numeric data type. Colon is the only valid separator and all colons prior to the first number are required.

APPENDIX I -- INDEX

This is an index of the terms and concepts used in this user guide. The numerical references are to page numbers. In cases in which one or two pages are especially relevant to the term or concept, those page numbers are underlined.

Module 17 ..... has not been indexed

Modules 41 ff ..... have not been indexed

Special Symbols

A

E

I

APPENDIX X -- GUIDE TO COMMON OPERATIONS

Operation	Command
Copying, of files	COPY
Creation, of abstracts	ABSTRACT
of data elements	ALLOCATE
of personal data bases	STORE, SUBSTITUTE
of tables	EDIT
Deletion, of files	DELETE
of lines from table	DELETE
of personal data bases	DELETE
Display, of abstracts	ABSTRACT
of available processor names	TOC
of files	LIST
of mixed data formats	FORMAT
of names of available data bases	TOC
of names of outstanding jobs	TOC
of system news	NEWS
of system status	BREAK
of table of contents	TOC
of transaction log	DLOG
of X Processor attributes	ATTRIBUTES
Duplication, of lines in table	INSERT
of tables	COPY
Editing, of tables	EDIT
Execution, conditional	IF
iterative	DO LOOP
of sequence tables	AUTOMATIC, BATCH, SEMIAUTOMATIC
of X Processors	PERFORM
Initialization, of Active Work Area	CLEAR, GET
of data elements	ASSIGN
Insertion, of lines in table	INSERT
Merging, of tables	INSERT
Modification, of abstracts	ABSTRACT, CHANGE
of data elements	ASSIGN
of file names	RENAME
of tables	EDIT, CHANGE, INSERT, DELETE
Movement, of files	APPEND, GET, STORE, SUBSTITUTE
Renumbering of tables	NUMBER
Searching, for information in tables	CROSS REFERENCE, FIND
Sign off	OFF

APPENDIX Y -- INDEX TO COMMANDS

APPENDIX Z -- QUICK REFERENCE TO COMMANDS

The following are sample commands. They are intended to be typical, but do not exhaust the possible uses and parameters associated with the commands.

```
FDS SIGNON:HR,INIT-PD,INIT1
ABSTRACT,GPMP-XP
ABSTRACT,OMS2-IT,"INTERFACE TABLE FOR OMS-2 BURN"
ALLOCATE,NUMVAL-IT,TWODIMEN(21,4)-DE
APPEND,EXPL
APPEND,EXPL,OMS2-IT,RESULT
ASSIGN,NUMVAL=20;
ASSIGN,TWODIMEN(21,I)=26.2*ONEDIMEN(I)'I=2,4;
ATTRIBUTES,GPMP-XP
ATTRIBUTES,GPMP-XP,APSOPT,SUMTABLE
AUTOMATIC,TRAJECT(20-230)
BATCH,EXPL-PD,TRAJECT
BREAK
CHANGE,TRAJECT-ST,OMS2,OMS2A
CLEAR
COPY,OMS2-IT,OMS2A
CROSS REFERENCE,-ST,OMS2-IT
DELETE,EXPL-PD
DELETE,(20-40)
DLOG-P
DO LOOP,WHILE,I<10;
   (statements)
LOOP END
DO LOOP,UNTIL,I>=10;
   (statements)
LOOP END
EDIT,,OMS2B-IT,GPMP
EDIT,,TRAJECT-ST
EXIT
FIND,TRAJECT-ST,OMS2
FIND,(40-200),X+16
FORMAT,M31,M33
GET,EXPL-PD
GET,!!FLIGHT-MD,-IT
IF,I<10;
   (statements)
ENDIF
IF,I<10;
   (statements)
ELSE
   (statements)
ENDIF
INSERT,TRAJECT(50-100)-ST,20
LIST-P,TRAJECT-ST
MESSAGE,AB,"I'VE FINISHED THE OMS-2 TABLE"
MODE,I
NEWS
NOTE,"UPPER STAGE MANEUVERS"
NUMBER
OFF
PERFORM,GPMP,OMS2
RENAME,OMS2-IT,OMS2A
SEMIAUTOMATIC,TRAJECT
STORE,EXPL
STORE,EXPL,TRAJECT-ST
SUBSTITUTE,EXPL
SUBSTITUTE,EXPL,TRAJECT-ST
TOC
TOC-P,,EXPL-PD
```

NASA-JSC